

2012

An Analysis of Data Quality Defects in Podcasting Systems

Thomas A. Mis
Cleveland State University

Follow this and additional works at: <https://engagedscholarship.csuohio.edu/etdarchive>

 Part of the [Electrical and Computer Engineering Commons](#)

How does access to this work benefit you? Let us know!

Recommended Citation

Mis, Thomas A., "An Analysis of Data Quality Defects in Podcasting Systems" (2012). *ETD Archive*. 737.
<https://engagedscholarship.csuohio.edu/etdarchive/737>

This Thesis is brought to you for free and open access by EngagedScholarship@CSU. It has been accepted for inclusion in ETD Archive by an authorized administrator of EngagedScholarship@CSU. For more information, please contact library.es@csuohio.edu.

**AN ANALYSIS OF DATA QUALITY DEFECTS IN
PODCASTING SYSTEMS**

THOMAS MIS

Bachelor of Science in Computer Science

John Carroll University

May, 1999

submitted in partial fulfillment of the requirements for the degree

MASTERS OF SCIENCE IN SOFTWARE ENGINEERING

at the

CLEVELAND STATE UNIVERSITY

December 2012

This thesis has been approved for the
Department of **ELECTRICAL AND COMPUTER ENGINEERING**
and the College of Graduate Studies by

Thesis Committee Chairperson, Dr. Nigamanth Sridhar

Department/Date

Dr. Yongjian Fu

Department/Date

Dr. Wenbing Zhao

Department/Date

To my parents...

ACKNOWLEDGMENTS

First and foremost I would like to thank Dr. Nigamanth Sridhar for his willingness to work with me across oceans and times zones. Without his guidance and seemingly infinite amount of patience this thesis would never have become a reality. Thank you to Scott Darpel, Maciej Zborowski and all of the students of the Industrial Space Systems Lab for making my time at Cleveland State memorable. Thank you Chris Paladino for starting a Podcast that turned into something truly special, and allowing me to become a part of it. Thank you to Mitch Gitelman for plucking me out of obscurity and providing me the chance to build a career at Microsoft. Thank you to Jon McCoy for the opportunities to grow as a software professional, and for encouraging me to take the time necessary to complete this thesis. Thank you to Sean Neumann and Adam Mollis for providing both motivation and distraction when I needed them most. Thank you to Chad Hantak, Chris Ivan, Keith Paladino, Bob Kopinsky and Phil Lock for being not necessarily the worst co-hosts in all of Podcasting. Finally I would like to thank both Phil LeMay and Heather Paladino for the graciousness and generosity they showed me while I completed my graduate coursework and started my professional career. Without their help, and the help of so many others, I would not have been able to accomplish any of my academic or career goals.

AN ANALYSIS OF DATA QUALITY DEFECTS IN PODCASTING SYSTEMS

THOMAS MIS

ABSTRACT

Podcasting has emerged as an asynchronous delay-tolerant method for the distribution of multimedia files through a network. Although podcasting has become a popular Internet application, users encounter frequent information quality problems in podcasting systems. To better understand the severity of these quality problems, we have applied the Total Data Quality Management methodology to podcasting. Through the application of this methodology we have quantified the data quality problems inherent within podcasting metadata, and performed an analysis that maps specific metadata defects to failures in popular commercial podcasting platforms. Furthermore, we extracted the Really Simple Syndication (RSS) feeds from the iTunes catalog for the purpose of performing the most comprehensive measurement of podcasting metadata to date. From these findings we attempted to improve the quality of podcasting data through the creation of a metadata validation tool — PodCop. PodCop extends existing RSS validation tools and encapsulates validation rules specific to the context of podcasting. We believe PodCop is the first attempt at improving the overall health of the podcasting ecosystem.

TABLE OF CONTENTS

	Page
ACKNOWLEDGMENTS	iv
ABSTRACT	v
LIST OF TABLES	ix
LIST OF FIGURES	xii
CHAPTER	
I. INTRODUCTION	1
1.1 The Problem	2
1.2 The Thesis	4
1.3 The Solution Approach	5
1.4 The Contributions	6
II. DEFINE PODCASTING	8
2.1 Podcast Characteristics	9
2.2 Podcast Architecture	11
2.2.1 Media Production	11
2.2.2 Publishing	13
2.2.3 Cataloging	15
2.2.4 Consumption	17
2.3 Quality Requirements	20
2.3.1 Correctness	20
2.3.2 Uniqueness	21
2.3.3 Platform Adherence	22
2.3.4 Chronology	22

2.3.5	Performance	22
III.	MEASURE PODCASTING	24
3.1	Validation Service	24
3.2	Validator Project	27
3.3	PodBot	28
3.3.1	Invoking Feed Validator	30
3.3.2	SyndicationFeed Class	30
3.3.3	Design Pitfalls	31
3.4	Measurements	32
3.4.1	The Data Set	34
3.4.2	Categories	34
3.4.3	Popularity	36
3.4.4	Metrics	38
IV.	ANALYZE PODCASTING	46
4.1	Quality Problems	46
4.1.1	Correctness Problem	47
4.1.2	Uniqueness Problem	49
4.1.3	Platform Adherence Problems	52
4.1.4	Chronology Problems	54
4.1.5	Performance Problems	55
4.2	Analysis Conclusions	56
V.	IMPROVE PODCASTING	57
5.1	PodCop Overview	58
5.1.1	Enforcing Correctness	59
5.1.2	Enforcing Uniqueness	61
5.1.3	Enforcing Chronology	61

5.2	PodCop Results	61
5.2.1	Random Podcasts	63
5.2.2	Popular Podcasts	63
VI.	RELATED WORK	66
6.1	Podcasting in Education	66
6.2	Podcasting in Mobile Networks	67
6.3	Podcast Metrics	69
6.4	Podcast Search	69
6.5	Podcast Consumption	70
6.6	Podcast Applications	71
VII.	CONCLUSION AND FUTURE WORK	72
7.1	Future Work	74
	BIBLIOGRAPHY	76
	APPENDIX	83

LIST OF TABLES

Table		Page
1	Example encoding guidelines from Apple	13
2	Example encoding guidelines from Microsoft	14
3	The top domains for feed hosting	16
4	The iTunes namespace	17
5	Exceptions thrown from SyndicationFeed	33
6	The iTunes inputs to PodBot	34
7	Number of feeds in each iTunes All Subcategories	36
8	Duplicate feeds discovered in iTunes	37
9	Example of a Duplicate feed	38
10	Top 10 Most Popular Feeds in Arts	39
11	Validation Results for All podcasts	40
12	Valid Feeds by Category	40
13	Invalid Feeds by Category	41
14	Invalid Feeds by Popularity	41
15	Size of podcast Feeds in Bytes	43
16	Size of RSS Elements in Characters	43
17	Length of podcast Episodes (hh:mm:ss)	44
18	Size of podcast Episodes (Bytes)	44
19	Number of Episodes per podcast Series	44
20	Top 5 Episode Format Types	44
21	Episode Sizes by Format	44
22	Top 5 XML Namespaces	44

23	Top 5 Languages	45
24	Top 5 RSS Authoring Tools	45
25	The Top 10 RSS 2.0 Errors	48
26	Examples of GUIDs Discovered by PodBot	50
27	Top Unsupported Formats Found by PodBot	53
28	PodCop Runtimes	59
29	PodCop Failure Rate for Individual Rules	62
30	Overall Failure Rate for All Podcasts	62
31	PodCop Results from Random Podcast Feed Sets	64
32	PodCop Results from Popular Podcast Feed Set	65
33	RSS 2.0 Channel Elements	92
34	The Top 50 RSS 2.0 Violations in Podcast Feeds	95
35	The RSS 2.0 Item Elements	96
36	The Top 30 Domains Hosting Podcast Feeds	97
37	The Top 10 Most Popular Arts Podcasts	98
38	The Top 10 Most Popular Business Podcasts	98
39	The Top 10 Most Popular Comedy Podcasts	99
40	The Top 10 Most Popular Education Podcasts	99
41	The Top 10 Most Popular Games & Hobbies Podcasts	99
42	The Top 10 Most Popular Government & Organizations Podcasts	100
43	The Top 10 Most Popular Health Podcasts	100
44	The Top 10 Most Popular Kids & Family Podcasts	101
45	The Top 10 Most Popular Music Podcasts	101
46	The Top 10 Most Popular News & Politics Podcasts	102
47	The Top 10 Most Popular Religion & Spirituality Podcasts	102

48	The Top 10 Most Popular Science & Medicine Podcasts	103
49	The Top 10 Most Popular Society & Culture Podcasts	103
50	The Top 10 Most Popular Sports & Recreation Podcasts	104
51	The Top 10 Most Popular Technology Podcasts	104
52	The Top 10 Most Popular TV & Film Podcasts	105
53	The Top 30 Podcast Episode Formats	106
54	Sizes of the Top 30 Podcast Formats	107
55	The Top 40 Podcast XML Namespaces	108
56	The Top 40 Podcast Languages	109
57	The Top 40 RSS 2.0 Authoring Tools	110

LIST OF FIGURES

Figure		Page
1	Viewing audio in Adobe Audition	12
2	Example of an RSS 2.0 Feed	15
3	The iTunes podcast catalog	16
4	The Zune podcast marketplace	18
5	The Zune podcast collection	19
6	The W3C Feed Validation Service	25
7	Example of Validation Failure	26
8	Invoking the Feed Validator	27
9	The PodBot Flowchart	29
10	SyndicationFeed Example	31
11	SyndicationLink Example	31
12	Example iTunes Catalog Entry	35
13	Example of Unresolved URI in Zune	48
14	Example of Uniqueness Bug in Zune	51
15	Example of Platform Adherence Bug in Zune	53
16	Example of Chronology Problem in Zune	54

CHAPTER I

INTRODUCTION

Podcasting has emerged as an asynchronous delay-tolerant method for the distribution of multimedia files through a network. Although podcasting has found wide adoption on mobile entertainment and phone platforms, it is not a technology introduced or supported by a single commercial software vendor, nor is it a technology standard governed by a standardization body such as the W3C or IETF. Furthermore, from a data quality perspective, podcasting can be viewed as a heterogeneous distributed database of multimedia files and the metadata that describes the various attributes of each file. Despite such complexity, and without the benefit of a formal governing organization, podcasting has grown to become a staple Internet technology used by nearly 70 million Americans in 2010 [16].

From the perspective of an end user, podcasting is an alternative to streaming media that allows for the consumption of audio and video files while disconnected from the Internet [14]. Users have access to a diverse set of entertainment and informational content provided by a wide variety of academic, governmental, and commercial organizations [37] such as Stanford University, Walt Disney, and the United States

Department of State. However it is the amateur user generated content community that first adopted podcasting, and the majority of multimedia content distributed through podcasting channels has been created by non-professional broadcasters. Furthermore, these collections of amateur produced content are not maintained by computer scientists or professional software engineers, hence the quality of these data sets is potentially problematic.

The research community has begun investigating aspects of podcasting and the utility of the podcasting distribution model in various contexts. Attempts have been made to model podcasting traffic through the Internet [4], and investigate the feasibility of distributing podcasts through ad hoc mobile networks [2,17,24,28,46]. There has been a desire to understand what makes a podcast popular, and to build systems that predict popularity based on programmatic inspection of podcasting data [42,43]. Network models have used popularity to help optimize passing podcasts through ad hoc mobile networks where nodes may not be in contact for prolonged periods of time [20]. The research community has shown much interest in measuring the effectiveness of podcasting as both a replacement for and a supplement to traditional classroom lectures [6,22,27,35]. Finally researchers who have experimented with producing educational podcasts have shared their experiences and provided guidelines for other Computer Scientists who wish to develop their own podcast series [13,45,48].

1.1 The Problem

Software engineering practitioners employ a variety of techniques and strategies for reducing the number of software failures that occur within software systems. A well-established practice such as group code reviews and white box testing can be used to validate a software system conforms to a functional specification. Black box testing is a software quality assurance strategy used to validate whether a software

system conforms to a functional specification by exercising features without knowledge of its actual implementation. Conversely white box testing uses knowledge of the implementation, often the source code itself, to create tests to validate a system. Development methods by which software engineers author code can also be employed to reduce the rate of software failures. Agile processes such as test driven development promote early testing of software components. This development methodology requires test cases to be authored before coding begins, and any newly coded software components are not considered complete until these test cases can be successfully exercised. Scrum and Extreme Programming encourage high quality by emphasizing the importance of testing throughout the software lifecycle rather than a process that only happens at the end, such as it would in the waterfall model.

These testing techniques and development processes focus on code. These evaluations are performed against static source code, or against code at runtime. Certainly architectural or syntactic defects in code can cause software to fail for a given purpose. However, code that could somehow be determined to be completely free of defects could still produce unacceptable outcomes if the inputs into the system contain defects. Reliable source code is insufficient to guarantee reliable outcomes for end users of software systems. Software can only produce error free output if the data into a system is itself free of defects. That is to say, reliable software will produce meaningful output for a user given a reliable set of data.

Zune is a brand of digital entertainment software and services from Microsoft. During the development of the Zune project this author employed the aforementioned techniques to validate the behavior of the Zune podcasting software. Functional testing was performed with sample podcast feeds that conformed to the Really Simple Syndication 2.0 standard (commonly referred to as RSS). Furthermore, negative testing was employed to validate error handling using podcast feeds that did not conform

to this standard. Although thorough code reviews and testing was performed against the podcast components, failures still emerged when real users consumed real podcast feeds. Similarly the iTunes entertainment client, the podcasting software from Apple, exhibited many of the exact same failures the Zune development team encountered when using real world feeds.

Analyzing the failures that occurred within both Zune and iTunes uncovered data quality problems that originated with the RSS feeds created by podcast producers. Given that podcasting is a multi-tiered Internet scale system, with no single governing organization, with tens of thousands of individual podcast producers contributing a constant stream of new media into the ecosystem, neither of these software development organization alone can impact the quality of the podcast data being produced. The World Wide Web Consortium has attempted to increase the quality of RSS by providing a validation service for statically analyzing an RSS feed for scheme violations. This service provided by the W3C is the only validation service available to podcast producers. However, it is now clear that passing RSS validation alone does not guarantee that a podcast feed is free of defects that can cause poor experiences for the end user. This is a significant problem facing the podcasting community. A better validation service is needed.

1.2 The Thesis

Although podcasting has become a popular Internet application, users encounter frequent information quality problems in podcasting systems. The existing validation tools available to podcast producers have not yielded a decrease in the number of defects encountered. Applying established data quality methodologies to podcasting will identify the source of many types of failures, and motivate a new set of data validation rules for podcasting feeds. These data validation rules are encap-

sulated in a next generation podcast validation tool. Use of this tool will decrease the number of software failures experienced by podcast consumers.

1.3 The Solution Approach

Data quality methodologies provide a framework for researchers and information professionals to investigate, understand, and improve the quality of data in complex information systems. Data quality processes in the context of software can be analogous to quality processes applied to physical materials in product manufacturing. In the context of manufacturing, engineers and quality professionals monitor and ensure raw materials are free of defects that could cause the end product to have flaws that are unacceptable to the customer. A key difference in this analogy however is that physical materials are consumed in the manufacturing process and provide a limit on the amount of low quality products produced, whereas low quality data will continuously impact the quality of the information systems consuming such data.

Data quality methodologies provide a framework for defining, measuring, analyzing, and improving data in information processing and database systems [26]. The research community has developed data quality methodologies where data serves as the raw materials into information processing systems [5]. A variety of data quality methodologies have been defined, many specific to a particular context or technology, such as biometrics [10] or relational databases [12]. Total Data Quality Management is a generalized data quality methodology that is designed to be applicable to a variety of contexts and systems [47]. Given this general purpose utility, TDQM will be applied to podcasting on the Internet for the purpose of understanding information quality defects and to identify opportunities for improvement. The podcast end user experience will be improved through the reduction of failures encountered by adhering to this formal data quality process to improve podcasting data.

Improving data requires an understanding of how the data is produced and distributed through the system. Therefore, the data quality approach will first define the components and stakeholders of the Internet wide podcasting ecosystem. Each data component will be inspected and documented in isolation and in relation to how it is consumed for the benefit of the end user. At the end of this phase podcasting will be a well understood and documented process. Furthermore, the quality requirements for each component will be defined.

Next the components of the podcasting system will be programmatically measured and analyzed. These phases will provide insight into the quantitative qualities of the podcasting ecosystem. Specifically, an automated system will be introduced to measure the quality of the distributed podcasting database. This system will organize the measurements and allow for researchers to analyze the data for trends. These trends will inform the creation of a podcast static analyzer for the purpose of evaluating podcast components for data quality failures.

1.4 The Contributions

This work provides value to a variety of stakeholders in the podcasting community. The net benefit of each of these contributions is to improve the quality of the user experience, such that podcast consumers can enjoy entertainment and informational content without encountering failures introduced from data quality problems. Podcast producers benefit from being able to easily identify potential quality defects that existing validation tools miss. And finally, the research community benefits from the most comprehensive modeling of the largest podcast sample set to date.

Specifically this work produced the following tangible contributions:

- The design and implementation of an automated web crawler for acquiring quality metrics on podcast feeds and media.

- The design and implementation of a validation tool for discovering podcasting specific quality defects in valid RSS feeds.
- The creation of a comprehensive model of podcasting characteristics to inform future research.

CHAPTER II

DEFINE PODCASTING

This project borrows upon the principles defined by the Total Data Quality Management methodology [47] for the purpose of improving the quality of podcasting. The TDQM methodology prescribes four activities an organization must perform to improve the information product produced by any given information system. The first of these activities is to rigorously define the various aspects of an information product, how the information product is produced, and the people involved in the consumption and production. This definition phase forces data quality professionals to systematically gain an understanding of the data to be improved. Therefore, in order to understand the aspects of podcasting that are to be improved, this chapter will define in detail the various characteristics of podcasting data, explore the architecture of podcasting systems, and identify the quality requirements that determine whether podcasting data can be considered “fit for use.”

It should be noted that this phase focuses on podcasts as data into an information manufacturing system, with particular focus on the quality attributes viewed as important to the end podcast consumer. The novelty of this project, and where

this project contributes value back to the podcasting community, comes from this focus on data rather than on software. That is to say, traditionally quality assurance professionals employed at software vendors that distribute podcasting systems apply various software testing techniques such as black box testing, white box testing, and fuzz testing to ensure software systems are of a particular quality [36]. The quality aspects of software is a well-researched area, and vendors such as Microsoft apply software testing techniques to ensure software systems are secure, accurate, and resistant to failure from poorly curated data [36]. As we shall see, although podcasting software systems may be well tested, podcasts distributed to users can be of low quality due to low quality inputs. podcasting suffers from a classic “garbage-in-garbage out” problem. Therefore, this project is the first of its kind that attempts to improve the data that serves as input into podcasting systems.

2.1 Podcast Characteristics

From the perspective of the consumer, podcasting is yet another form of serialized media. One could argue that podcasting has seen a degree of popularity due to the fact that many media consumers are already comfortable with the concept of media being distributed in a periodic manner. For example, those with a subscription to the newspaper The New York Times already expect a physical copy to be delivered to their doorstep each morning. The subscriber can read the newspaper as soon as they receive it, or they can read the newspaper at a later time, perhaps while riding the subway on their way to work during their morning commute. Similarly, digital media consumers can receive podcasts on regular intervals but can consume the podcasts at a later time. The newspaper analogy remains applicable with the advent of smart phones and portable media devices, as the podcast subscriber can consume the podcasts on the very same subway car as the New York Times subscriber. Later in

this chapter we will formally define the podcasting terms and concepts that will be referred to throughout this thesis. For now, let us continue to view podcasting from the perspective of a non-computer scientist in order to build an understanding of how podcasts are typically discovered and consumed.

A vernacular has evolved to describe the various attributes and characteristics of podcasting. Unfortunately, many of these terms contain the word podcast which itself has become a context dependent term. The term podcast can refer to an individual episode within a series, or it can refer to an entire series. Furthermore, podcasting is used to refer to the action of recording a podcast, or it can be used as a universal term to describe the entire collection of content and software that exists to build and distribute media. A list of terms is provided for reference.

- **Podcaster** is a human who publishes a podcast series. Hobbyist podcast producers generally managed the entire range of responsibilities necessary for publishing a podcast onto the Internet. These tasks can include: audio recording, media encoding, RSS authoring, and web hosting.
- **Podcast Feed** is an RSS 2.0 file hosted on a web server that is regularly updated by a podcast producer and consumed by podcasting systems.
- **Podcast Aggregator** is a software client that consumes multiple podcast feeds. Examples of popular commercial podcast aggregators include iTunes and Zune.
- **Podcast Series** is a general term used to describe the collection of podcast episode files listed within a podcast feed. However, it is often the case that only the most recent subset of episodes is actually enumerated within a feed.
- **Podcast Catalog** is an Internet service that contains a curated collection of links to podcast feeds. Examples of popular commercial podcast catalogs are the iTunes Store and Zune Marketplace.

2.2 Podcast Architecture

In the physical world manufacturing systems consume raw materials to produce finished products. In the context of Ford Motor Company, steel enters the manufacturing plant on one end, and a finished automobile is produced on the other. Industrial Engineers at Ford follow quality processes such as Six Sigma to reduce defects in automobiles by increasing the quality of raw materials entering the manufacturing plant. The TDQM methodology prescribes that software professionals identify their own information manufacturing systems. Therefore, this section will document the podcast manufacturing system. Digital audio and video are the raw materials that will be operated upon to produce a finished podcast.

2.2.1 Media Production

Instructional guidelines for producing podcasts in an academic context have been published [1, 13, 23, 41, 45, 48]. These guidelines focus on capturing university lectures into digital audio and video files. The systems range from elaborate automated lecture recording systems that are integrated directly into the classroom, to cost-conscious guidelines for purchasing off-the-shelf recording equipment for home use. Regardless of budget, the podcast information manufacturing system begins with the recording of audio or video.

Professionally produced shows have emerged in the podcasting space, examples of such shows are those produced by This Week In Tech and the Revision3 networks. These podcasting networks hire professional audio and video engineers to record and edit media for the purpose of achieving production values that equal traditional broadcast television. This author's own personal podcast [19] however is representative of the opposite end of the podcasting spectrum where the "on-air" personalities also serve as producers and audio engineers. In this scenario it is common to use widely

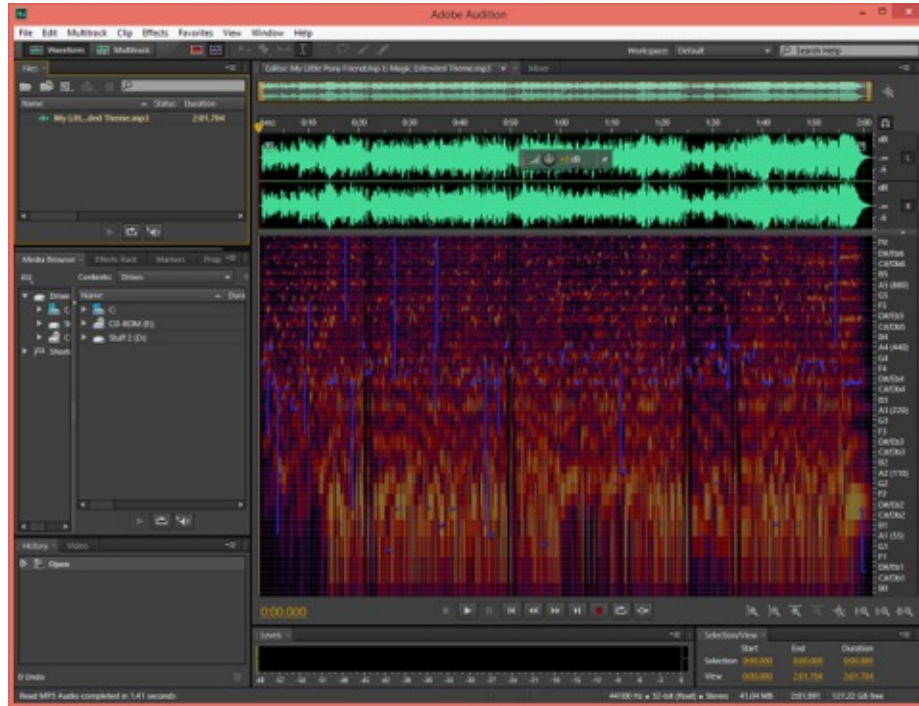


Figure 1: Viewing audio in Adobe Audition

available software such as Skype for facilitating the recording of multiple co-hosts residing in different geographies, and Adobe Audition (see Figure 1) to capture each audio stream for easy post-production editing and encoding.

It is at this point that the individual podcasters must encode their recorded media into a file appropriate for distribution through the Internet. As we shall see in Chapter 3, audio remains the predominate media type for podcasting content. It is also here where the first kinds of data defects are entered into the podcast information manufacturing system as not all podcasters have a good understanding of the tradeoffs between the various media encoding formats. Software vendors such as Microsoft and Apple provide podcasters with guidelines [3,29] for how to optimize encoding for playback on their platforms. Unfortunately platforms provided by these two companies often support a different set of codecs, thus confusing the matter further for both podcasters and podcast subscribers. Tables 1 and 2 provide examples of video encoding guidelines from Apple and Microsoft for a few select devices. Notice

Device	Encoding Guideline
iPod Touch	“H.264 video up to 720p, 30 frames per second, Main Profile level 3.1 with AAC-LC audio up to 160 Kbps, 48kHz, stereo audio in .m4v, .mp4, and .mov file formats.”
iPod Touch	“MPEG-4 video, up to 2.5 Mbps, 640 by 480 pixels, 30 frames per second, Simple Profile with AAC-LC audio up to 160 Kbps per channel, 48kHz, stereo audio in .m4v, .mp4, and .mov file formats.”
iPod Touch	“Motion JPEG (M-JPEG) up to 35 Mbps, 1280 by 720 pixels, 30 frames per second, audio in ulaw, PCM stereo audio in .avi file format.”
iPhone 3GS	“H.264 video, up to 1.5 Mbps, 640 x 480, 30 frames per sec., Low-Complexity version of the Baseline Profile with AAC-LC audio up to 160 kbps, 48 Khz, stereo audio in .m4v, .mp4, and .mov file formats.”
iPhone 3GS	“H.264 video, up to 768 kbps, 320 x 240, 30 frames per sec., Baseline Profile up to Level 1.3 with AAC-LC audio up to 160 kbps, 48 Khz, stereo audio in .m4v, .mp4, and .mov file formats.”
iPhone 3GS	“MPEG-4 video, up to 2.5 Mbps, 640 x 480, 30 frames per sec., Simple Profile with AAC-LC audio up to 160 kbps, 48 Khz, stereo audio in .m4v, .mp4, and .mov file formats.”

Table 1: Example encoding guidelines from Apple

that encoding H.264 video at a resolution of 720x480 will allow for playback on the Zune and iPod Touch, but that resolution is not supported on the iPhone 3GS. Podcast producers must be aware of these discrepancies.

2.2.2 Publishing

Once media files are edited and encoded into the desired format, the individual podcasters are responsible for hosting the media files on a web server. Podcasters within academic departments may seek hosting from their university resources, but independent podcasters generally must purchase web hosting from a commercial hosting company. Given that media files are larger than HTML content, the cost of purchasing hosting from a commercial company increases at a faster rate as a podcast series becomes popular as compared to a web site distributing text. The media files that have been uploaded onto the World Wide Web are now accessible through

Device	Encoding Guideline
Zune 8GB	Windows Media Video Simple Profile (.wmv) - up to 320x240, 10fps and 1.5 Mbps.
Zune 8GB	Windows Media Video Main Profile (.wmv) - up to 720x480, 30fps and 3 Mbps.
Zune 8GB	H.264 baseline profile video with AAC audio (.mv4, .mp4) - up to 720x480, 30fps and 2.5 Mbps.
Zune 8GB	MPEG4 Part 2 simple profile video with AAC audio (.mv4, .mp4) - up to 720x480, 30fps and 2.5 Mbps

Table 2: Example encoding guidelines from Microsoft

hyperlinking. In fact, it is common for these shows to be linked to from an HTML file.

What makes a media file a podcast however is when it is listed as an entry in an RSS 2.0 feed. When an RSS 2.0 feed contains media content we generally refer to it as a podcast feed. Creating and hosting the podcast feed is also the responsibility of the individual podcasters. Figure 2 contains an example of a podcast feed from National Public Radio [33]. Extraneous namespace elements have been removed to simplify the example.

The RSS 2.0 documentation is maintained by the Berkman Center for Internet & Society at Harvard Law School [8]. A description of each element in the RSS scheme is provided by the Berkman Center, along with usage examples. Tables 33 and 35 contain a complete copy of these descriptions and example element values.

Commercial feed hosting services have emerged to provide podcasters with free RSS hosting. Table 3 lists the most popular hosting services discovered during our investigation that is described in the next chapter. For now, note that Feedburner (a Google subsidiary) is by far the most popular feed host with 24% of all feeds, followed by Libsyn at 5%. Both of these companies publically market themselves as services to help independent content creators monetize the media they produce. Feedburner and Libsyn attempt to act as intermediaries between agencies seeking to advertise

```

<?xml version="1.0" encoding="UTF-8" ?>
<rss version="2.0">
  <channel>
    <title>NPR: Hourly News Summary Podcast</title>
    <link>http://www.npr.org/templates/topics/topic.php?topicId=1001</link>
    <description><![CDATA[ Five minutes of NPR news, updated hourly.]]></description>
    <copyright>Copyright 2007 NPR</copyright>
    <generator>NPR API RSS Generator 0.94</generator>
    <language>en-us</language>
    <image>
      <url>http://media.npr.org/images/podcasts/thumbnail/npr\_hourlynews\_image\_75.jpg</url>
      <title>NPR: Hourly News Summary Podcast</title>
      <link>http://www.npr.org/templates/topics/topic.php?topicId=1001</link>
    </image>
    <lastBuildDate>Sun, 09 Sep 2012 20:19:15 -0400</lastBuildDate>
    <item>
      <title>NPR News: 09-09-2012 8PM ET</title>
      <description><![CDATA[ NPR News: 09-09-2012 8PM ET]]></description>
      <pubDate>Sun, 09 Sep 2012 20:19:15 -0400</pubDate>
      <link>http://www.npr.org/templates/topics/topic.php?topicId=1001</link>
      <guid>http://podcastdownload.npr.org/npr.mp3</guid>
      <enclosure url="http://podcastdownload.npr.org/npr.mp3" length="2313489" type="audio/mpeg" />
    </item>
  </channel>
</rss>

```

Figure 2: Example of an RSS 2.0 Feed

and independent podcasters. It is not clear how successful these efforts to monetize podcasting have been.

2.2.3 Cataloging

The creation of media and the hosting of media files are the responsibility of the individual podcast producers. This thesis has discovered 19,849 unique domains hosting podcast feeds (Table 36 lists the top domains). With 72,786 unique podcast feeds spread across these nineteen thousand hosts, finding an appealing show can be a difficult problem for users. Attempts have been made to provide web based directories of podcast feeds. A web portal cited in the existing podcast researcher [4] is Odeo.com. However, as of this writing, Odeo.com no longer functions as a podcast directory. Fortunately podcast cataloging services have emerged from Microsoft and Apple.

Microsoft and Apple provide cataloging services to ease the podcast discovery

Category	Domain	Feed count	Feed %
.com	feedburner.com	17616	24.25%
	libsyn.com	3753	5.17%
	podbean.com	2481	3.42%
	podomatic.com	1898	2.61%
	blip.tv	1583	2.18%
.gov	nasa.gov	50	
	nps.gov	39	
	cdc.gov	30	
	senate.gov	22	
	nih.gov	14	
.edu	wisc.edu	24	
	si.edu	20	
	ufl.edu	19	
	umich.edu	17	
	umn.edu	16	

Table 3: The top domains for feed hosting

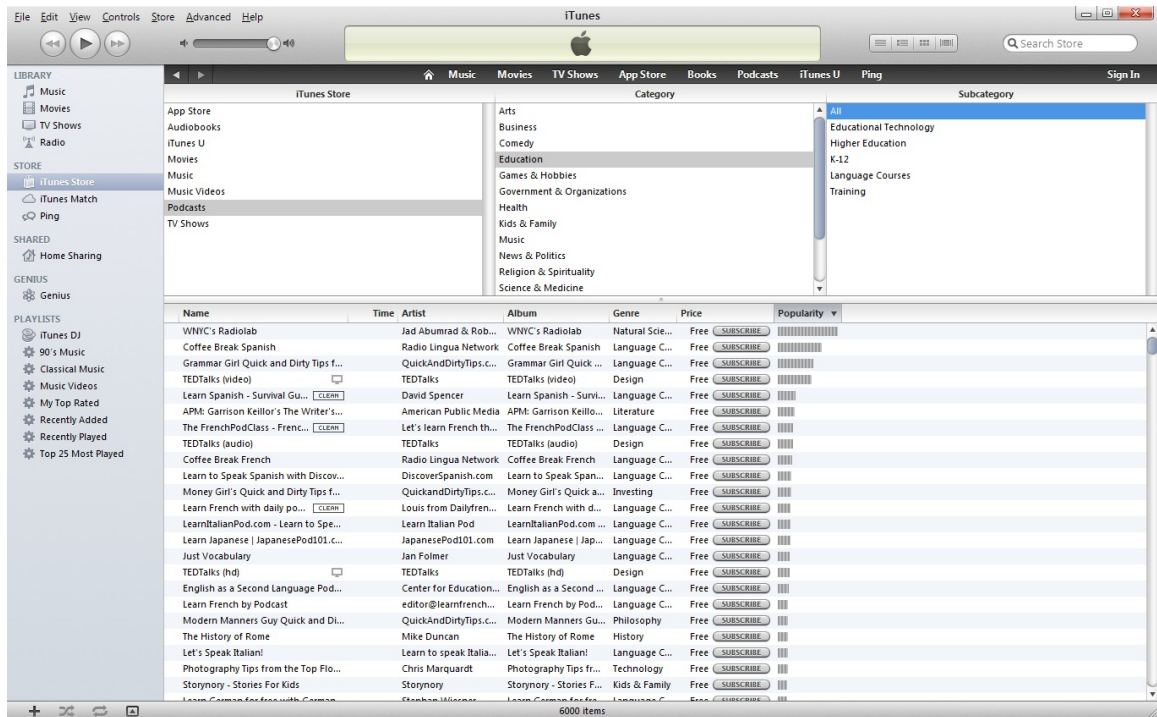


Figure 3: The iTunes podcast catalog

Element	Description
<code>itunes:author</code>	Artist column in iTunes.
<code>itunes:block</code>	Prevent an episode or podcast from appearing in iTunes.
<code>itunes:category</code>	Category column and in iTunes Store Browse.
<code>itunes:image</code>	Album art displayed in iTunes.
<code>itunes:duration</code>	Time column in iTunes.
<code>itunes:explicit</code>	Parental advisory graphic in iTunes.
<code>itunes:isClosedCaptioned</code>	Closed Caption graphic in iTunes.
<code>itunes:order</code>	Override the order of episodes in the store.
<code>itunes:complete</code>	Indicates completion of podcasts; no more episodes.
<code>itunes:keywords</code>	Not visible but can be searched.
<code>itunes:new-feed-url</code>	Not visible, used to inform iTunes of new feed URL.
<code>itunes:owner</code>	Not visible, used for contact only.
<code>itunes:subtitle</code>	Description column in iTunes.
<code>itunes:summary</code>	The More Info field in iTunes.

Table 4: The iTunes namespace

process for consumers of their Zune and iPod portable digital media devices. The data set for this work was acquired from the Apple iTunes catalog service (see Figure 3). An overview of the findings from an automated inspection of the iTunes podcast catalog is described in Chapter 3.

As a prerequisite for inclusion into the iTunes catalog, Apple mandates that podcasters adopt the iTunes namespace [3]. The iTunes namespace extends the RSS 2.0 specification with elements and attributes that describe attributes specific to podcasting. A brief overview of the iTunes namespace elements is provided in Table 4. These elements were inspected as part of the TDQM measurement phase.

2.2.4 Consumption

Portal media devices and mobile phones enable users to subscribe to podcasts from wireless networks. Although the portability of media devices provides for a

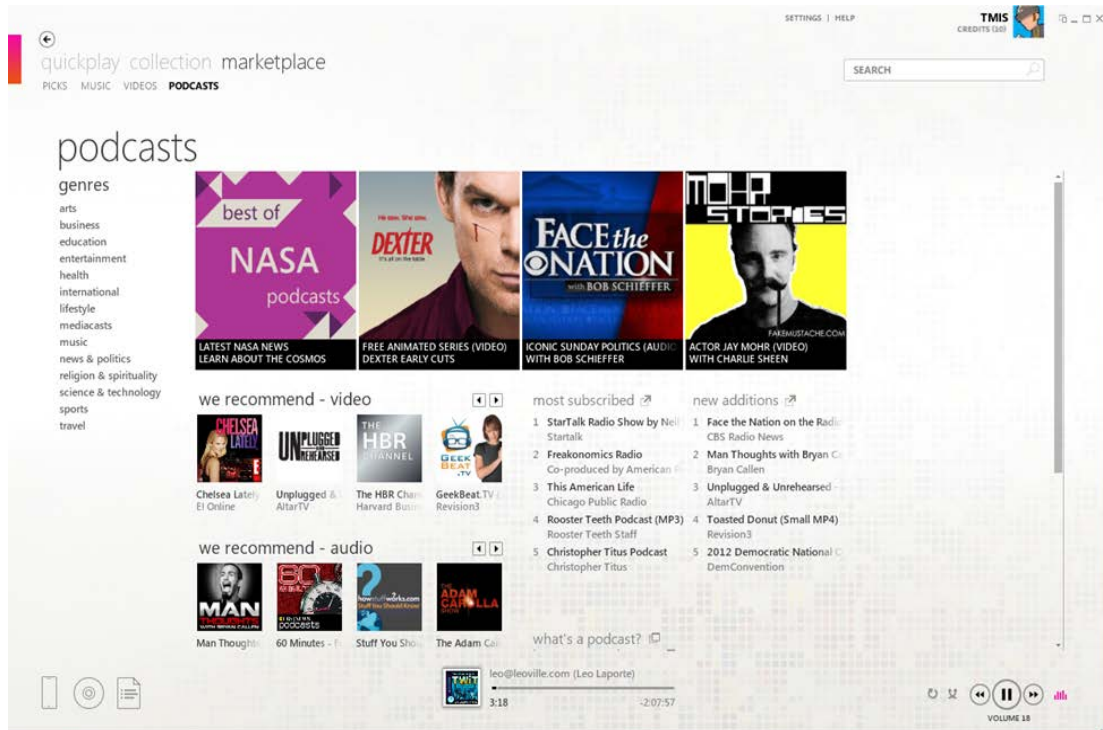


Figure 4: The Zune podcast marketplace

variety of media consumption scenarios, it has been found that the majority of podcast consumption occurs on personal computers. A survey of users found that 68% of audio podcasts and 77% of video podcasts are consumed from a computer rather than a portable device [16]. A survey of computer science students at Harvard University revealed that 71% of the students consumed podcasts at personal computers (29% on iPod devices) when lectures were made available in podcast form [27]. The PC is a popular platform for consuming podcasting content.

Two commercially available podcast aggregators for the PC platform are Apple iTunes and Microsoft Zune. To better understand podcast consumption scenarios on the PC, the Zune software client was inspected. The podcast functionality built into the Zune software enables podcast searching, subscribing, syncing to mobile device, and playback.

The Zune podcast marketplace (see Figure 4) provides a graphical user inter-

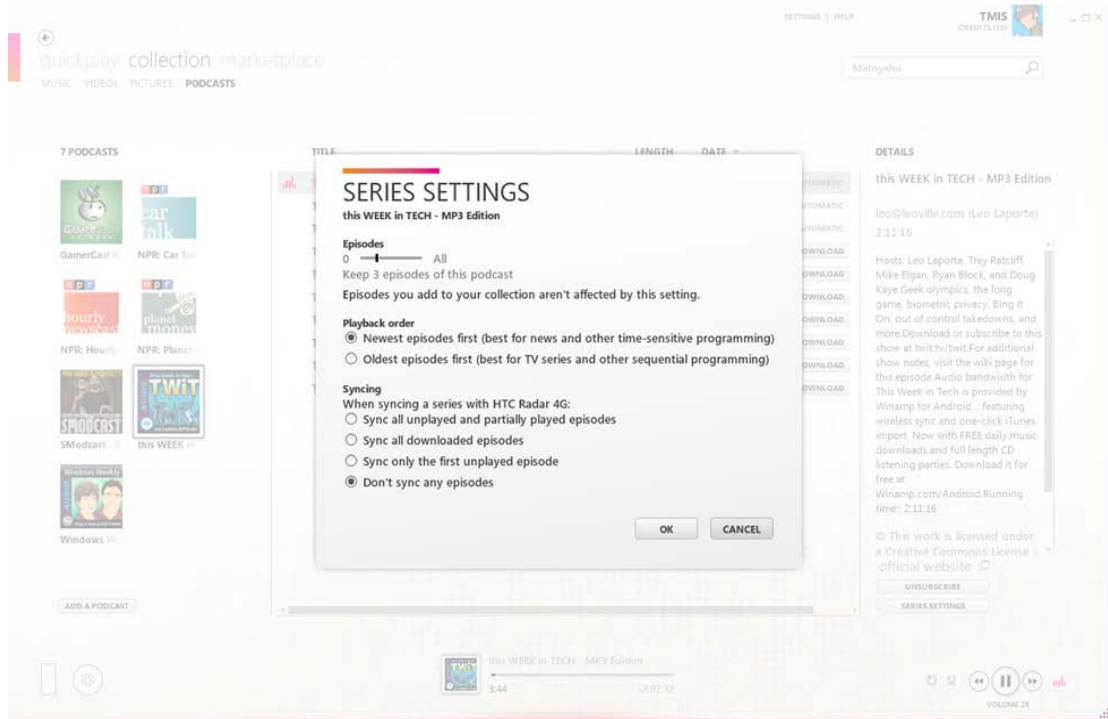


Figure 5: The Zune podcast collection

face around the podcast catalog web service. The marketplace organizes podcasts into distinct categories, such as: business, education, and travel. Viewing each category within the user interface causes the podcasts to be displayed in order of popularity. It is through this interface that users discover new podcasts series. Selecting a podcast within the marketplace displays a details page that exposes the metadata from the RSS feed for that show. The details page also provides a graphical method for subscribing to the podcast. Subscribing to a podcast causes the show to be added to the podcast collection.

The Zune podcast collection is a graphical user interface for managing podcast subscriptions and syncing podcasts to portable media devices. The collection interface (see Figure 5) provides a set of settings for creating rules that define the order a podcast series should be consumed, the rules for syncing new episodes to devices, and the amount of content to download and stored on the personal computer. Podcast

episodes that have been downloaded and stored on the personal computer can be played back from the collection view. Each time the Zune software is launched, it downloads the latest version of the RSS feed for each of the shows in the podcast collection. If new episodes are available, they are downloaded according to the download rules. The download progress is displayed in a graphical manner, and the download of podcast episodes can be manually paused or cancelled by the user.

2.3 Quality Requirements

Techniques have been developed [42, 43] to measure the quality of podcasts from the perspective of what makes a podcast popular with consumers. The quality requirements defined in this thesis however focus on reducing the source of information quality problems in podcasting systems. That is, this thesis is not concerned with the creation of professionally produced and entertaining media presentations. Rather this thesis is focused on ensuring that the data entered into podcasting systems is well-structured and free of defects that cause failures.

This project defines a collection of quality requirements that must be satisfied for a podcast to be considered free of defects. In adherence with the TDQM methodology the quality requirements are defined during the Definition Phase and specific examples of violating each of these requirements will be examined in the Analysis Phase. Each of the requirements is described below.

2.3.1 Correctness

The correctness requirement can apply to both the structure and content of podcasting data. A podcast feed that satisfies the correctness requirement must adhere to the RSS 2.0 specification and the values of the feed elements must be factually correct. Some examples of violating the correctness requirement can include: invalid

XML markup, providing URI values that do not resolve, and factually incorrect metadata values.

Validating metadata values can be a difficult computer science problem. It may not be possible to know whether a text field containing a description of a podcast episode is associated with the correct audio file within a podcast feed. There have been research efforts to programmatically analyze podcast episodes using speech-to-text systems [34].

Even if a reliable solution was developed, it may not be impactful to reducing quality problems. If the wrong text description is displayed within a podcast aggregator, it may lead the user to listen to an episode that they may not be interested in, but an incorrect description most likely will not cause the user to perceive the software system as being unreliable. Thus the fields that will be validated for correctness in the Improvement Phase will be those that can cause more serious failures in podcast aggregators. The size attribute of the enclosure element is one such field. If the actual media file size is different than the size denoted in the feed metadata, then podcasting systems may poor choices around file downloading and memory allocation.

2.3.2 Uniqueness

A podcast episode is unique if podcast aggregators are able to distinguish an episode from all other previously and currently published episodes in the series. Podcast aggregators regularly inspect podcast feeds for new episodes and changes to metadata. If a podcast aggregator cannot determine a new episode has been added to the feed, then the new episode will not be downloaded and the user will not be able to consume the episode. During the development of Zune, it was observed that changes were often made to the metadata that describes an existing podcast episode. An example of this scenario is a podcast episode that was published with

a spelling mistake in the title element of the feed; often podcasters would republish the feed with the corrected spelling. In this case, the podcast aggregators must be able to distinguish between an updated existing episode, and an entirely new episode. Failure to do so can cause the aggregators download the same show multiples time. This may not be a bad problem for a personal computer on a high speed Internet connection, but it can be a bad problem for a mobile phone on a data connection that incurs fees for high usage.

2.3.3 Platform Adherence

Popular vendors such as Apple and Microsoft publish guides that provide platform specific metadata format and media encoding recommendations to podcast producers. Examples of violating the property of platform adherence can include: playback failures from incorrectly encoded media, download failures from linking to non-acceptable content, and display failures from incorrectly formatted metadata.

2.3.4 Chronology

Each podcast episode has a temporal property that should reflect its proper chronological order within a series. Failure to satisfy this property causes podcast aggregators to display episodes in an unintended ordering.

2.3.5 Performance

This property can be platform specific. Users on a PC with a cable modem may not be sensitive to large feeds or media files. Users on mobile phones with limited or costly data plans may be very sensitive to large feeds and media files. Furthermore, having extreme values in a feed may cause aggregators to ignore and stop processing a feed. Violating this property may be considered a warning to borrow a concept from

language compilers. Examples of violating the performance property can include: large numbers of item elements in a feed that are truncated by aggregators, and large media files that may not be able to transfer over mobile data connection.

The initial phase performed by data quality professionals tasked with improving the quality of data manufacturing systems is to define the characteristics of the system, and the requirements for determining data is of high quality. In fulfillment of this phase, the characteristics of podcasting architecture and podcasting feeds have been described. An exploration of the Zune podcast aggregator was conducted. And the quality requirements of correctness, uniqueness, platform adherence, chronology, and performance have been established. The next phase of the TDQM methodology is to measure the data that is to be improved.

CHAPTER III

MEASURE PODCASTING

The TDQM methodology recommends that information quality metrics are defined and measured. These findings help quality professionals understand how the data into an information manufacturing system satisfies the data quality requirements developed during the definition phase. This chapter measures the various dimensions of podcast feeds. These measurements are the first to shine light on the scope of the data quality problems that exist within the medium.

3.1 Validation Service

The World Wide Web Consortium is the international organization that publishes standards and guidelines for developing World Wide Web technologies [49]. Although the RSS 2.0 specification itself is not a W3C recommendation, the organization does host an RSS 2.0 validation service (see Figure 6) on the w3c.org website [50]. This validation service can be used by authors of podcast feeds to identify any markup that violates a rule of the RSS 2.0 specification. Adherence to web stan-

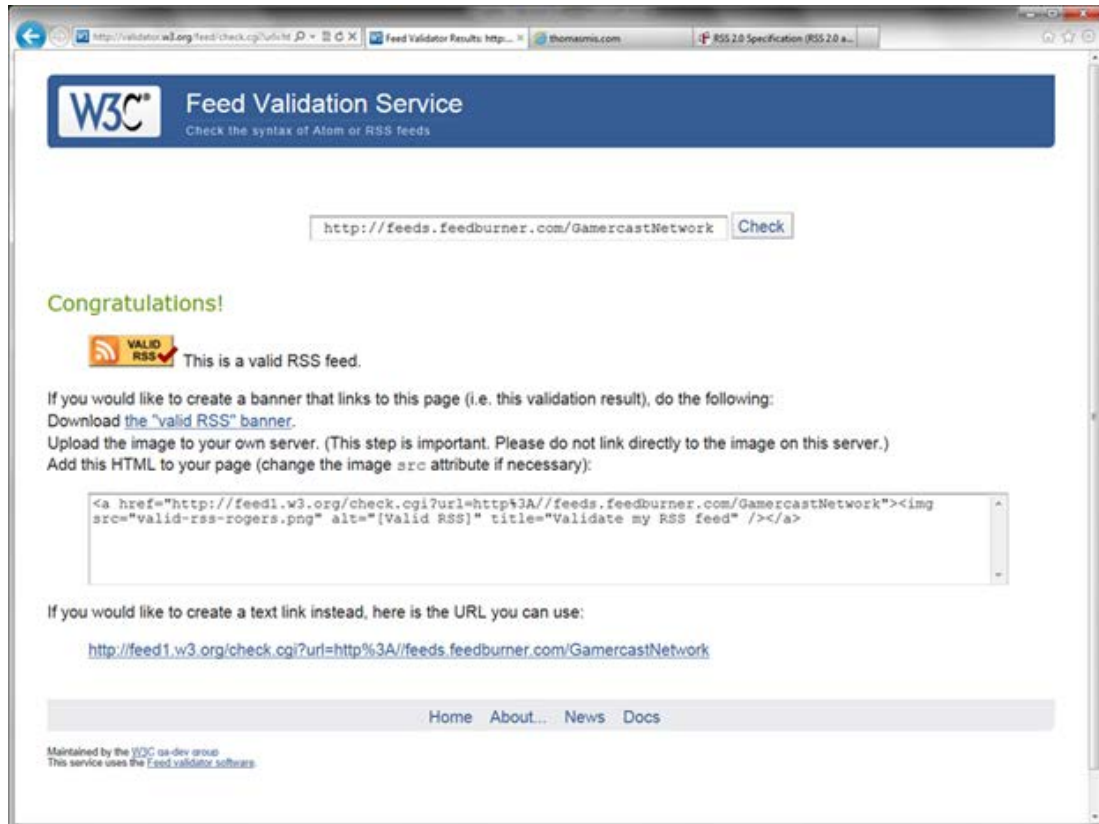


Figure 6: The W3C Feed Validation Service

dards, such as HTML 4.0 or XHTML 1.0, ensures that web content is accessible to the widest variety of client software such as Mozilla and Internet Explorer. Similarly, podcast feeds that adhere to the RSS 2.0 standard ensure they are accessible to a wide variety of podcast aggregators such as iTunes and Zune. However, passing RSS validation alone does not guarantee a podcast feed is free from defects that cause information quality problems.

With multiple versions (0.91, 0.92, and 2.0) [7] and with the acronym itself having had multiple meanings (RDF Site Summary, Rich Site Summary, and Really Simple Syndication), RSS has had something of a tumultuous history. The 2.0 version of the standard is currently maintained by the Berkman Center for Internet & Society at Harvard University [8] and the independent RSS Advisory Board [39]. The

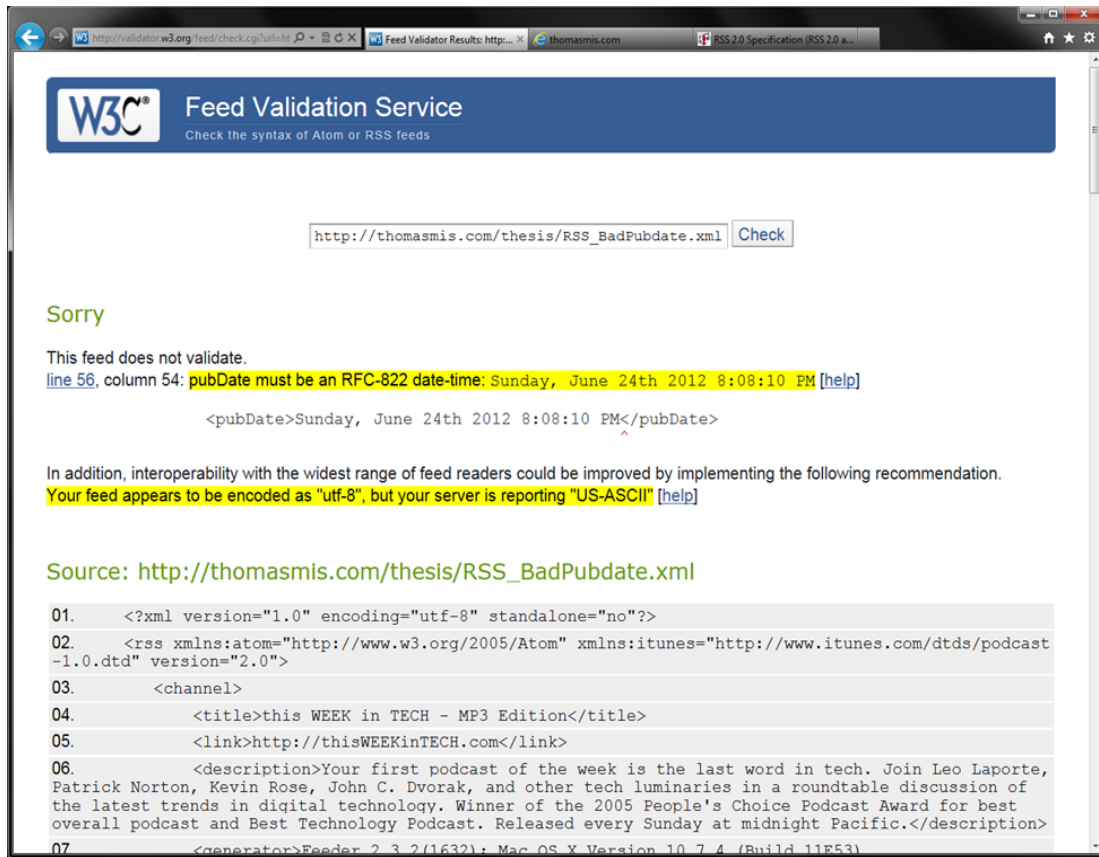
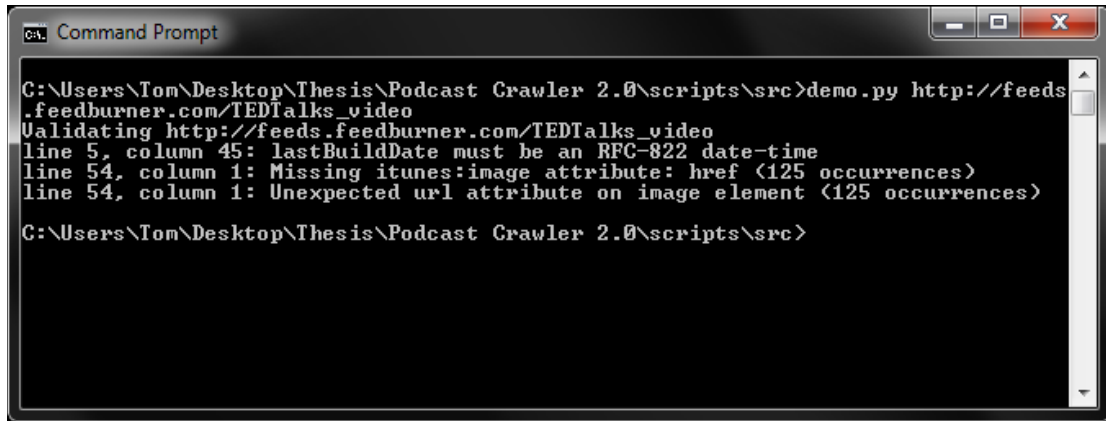


Figure 7: Example of Validation Failure

significance of the W3C providing a validation service for RSS 2.0 is that software developers and podcasters can have confidence that this version of RSS has been acknowledged by a reputable Internet standards organization. Thus, podcasting systems that support RSS 2.0 will not be vulnerable to rapid specification changes that cause compatibility failures for end users.

The primary method for validating a podcast feed is to submit the URI for the feed under test into the appropriate field on the W3C website (see Figure 7). Much like compiling C++ code, the amount of time necessary to complete the validation is a function of the size of the source file; in general the validation runtime was on the order of seconds. The result of performing the validation against a feed can either be a congratulatory message for submitting markup that adheres to the RSS



```
Command Prompt
C:\Users\Tom\Desktop\Thesis\Podcast Crawler 2.0\scripts\src>demo.py http://feeds.feedburner.com/TEDTalks_video
Validating http://feeds.feedburner.com/TEDTalks_video
line 5, column 45: lastBuildDate must be an RFC-822 date-time
line 54, column 1: Missing itunes:image attribute: href (125 occurrences)
line 54, column 1: Unexpected url attribute on image element (125 occurrences)
C:\Users\Tom\Desktop\Thesis\Podcast Crawler 2.0\scripts\src>
```

Figure 8: Invoking the Feed Validator

2.0 standard or, a set of error and warning messages. Error messages contain specific line numbers, and an explanation of how the markup at the specified location in the source file violates an RSS 2.0 rule.

3.2 Validator Project

The actual feed parsing and rule validation logic for the W3C service is performed remotely. This server side component is provided by the open source Feed Validator project [38] which distributes the software under the MIT software license. Both Apple and Microsoft refer content creators [3, 30] to the Feed Validator project as a prerequisite to catalog ingestion. The Feed Validator project makes a command line version (see Figure 8) of this service available as a tool for validating feeds without the web interface. A description of how this stand-alone version of the Feed Validator was leveraged for this investigation into the data quality of podcasting will be provided in the next section.

3.3 PodBot

PodBot is an Internet connected software agent responsible for finding and inspecting publicly accessible podcast feeds. The data quality measurements discovered during this thesis were acquired by unleashing these agents onto the Internet. Development of the PodBot software comprised a significant portion of the time budget for this project. Therefore in order to reduce the cost of future research into the various aspects of podcasting, we now discuss the PodBot web crawler architecture and design tradeoffs. Documenting the lessons learned during development and execution can help future podcast researchers avoid the pitfalls encountered here.

PodBot is responsible for acquiring, validating, and parsing podcast feeds. These activities are executed in a linear fashion for each feed. Execution of each activity is predicated on the success of the previous activity. Therefore a feed will only be validated if acquisition succeeded, and a feed will only be parsed if validation succeeded. Any activity that failed was logged, and an investigation of these failures is discussed in Section 4.1. Figure 9 provides a visual diagram of the PodBot discussion making tree.

The PodBot crawler was implemented to be single threaded. To achieve parallelism and thus decrease total runtime, individual instances of the PodBot crawler were run across multiple computers. To further simplify development, it was assumed that each podcast feed will only be visited by an instance of the crawler at most one time. The benefit of this assumption was that the PodBots themselves did not need to manage the complexity of coordinating crawls amongst multiple instances, as each instance was responsible for a unique subset of all podcast feeds. However, the drawback of this limitation is that this project will not benefit from learning how a podcast feed has changed over time.

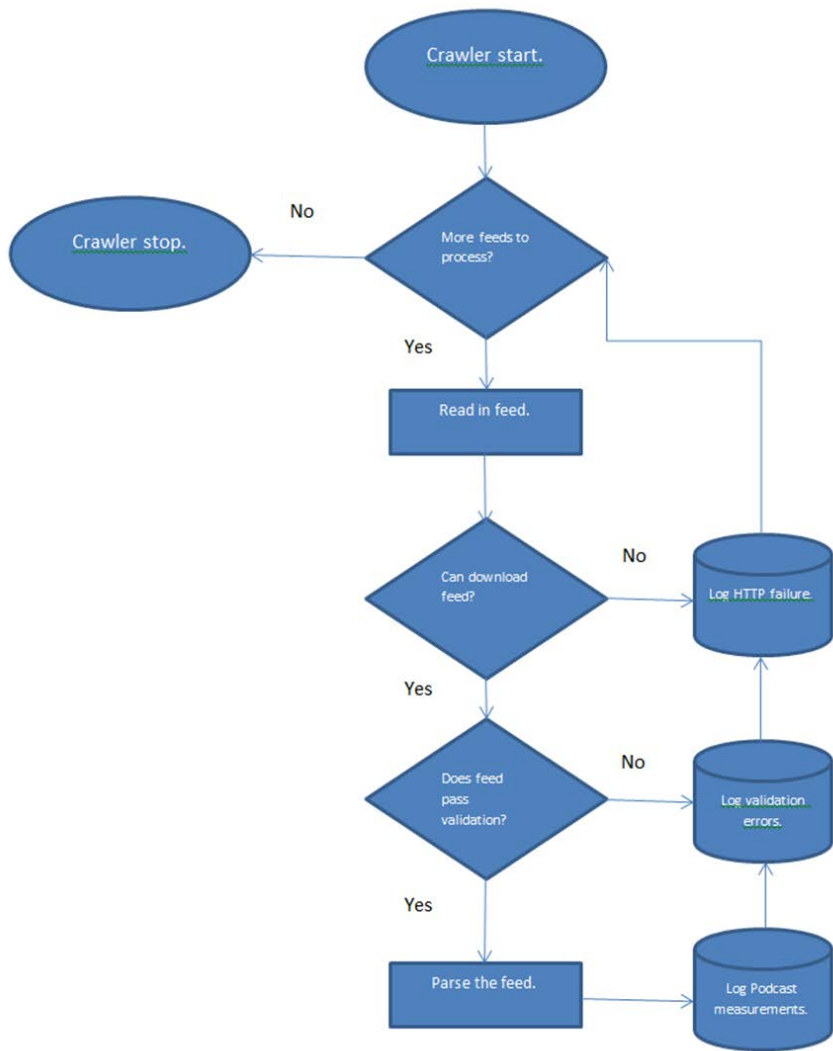


Figure 9: The PodBot Flowchart

3.3.1 Invoking Feed Validator

Each PodBot crawler spawns a separate process in order to perform the validation task. Rather than implement its own RSS 2.0 validation component, the crawler invokes the open source Feed Validator project to provide this functionality. This process separation was necessitated by the differing runtime dependences for each project. The PodBot project is built upon the Microsoft .NET runtime, and the Feed Validator project is built upon the Python runtime. Any validation failures discovered by the Feed Validator are captured and logged by PodBot, an investigation of these failures across all podcast feeds is discussed in Section 4.1.

3.3.2 SyndicationFeed Class

The Microsoft .NET framework was chosen as the platform for the crawler due to the inclusion of the SyndicationFeed class. This class was introduced with version 3.5 of the .NET framework [31]. An object of type SyndicationFeed represents an Atom 1.0 (an alternative XML format for syndicating web content) or RSS 2.0 feed. These objects have the capability to parse existing feeds into internal data structures, change the values in existing feeds, and create entirely new feeds. The PodBot project makes extensive use of the SyndicationFeed class for its ability to download and parse publicly available podcast feeds. Examples of instantiating the SyndicationFeed object for inspecting each field of a podcast feed is listed in Figure 10. The appendix contains a complete C# sample application that demonstrates using each of the public SyndicationFeed methods.

Perhaps what most distinguishes a podcast feed from news feeds containing only text data is the use of the optional enclosure element. It is the enclosure elements that contain links to the digital media files that comprise the episodes of a podcast series. Unfortunately parsing the enclosure elements of a podcast feed was

```

using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ServiceModel.Syndication;
using System.Xml;
using System.Xml.Linq;

namespace SyndicationExample
{
    class Program
    {
        static void Main(string[] args)
        {
            string feed = args[0];

            XmlReader xmlReader = XmlReader.Create(feed);
            SyndicationFeed syndicationFeed = SyndicationFeed.Load(xmlReader);

```

Figure 10: SyndicationFeed Example

```

XmlReader xmlReader = XmlReader.Create(feed);
SyndicationFeed syndicationFeed = SyndicationFeed.Load(xmlReader);

foreach (SyndicationItem item in syndicationFeed.Items)
{
    foreach (SyndicationLink link in item.Links)
    {
        if (link.RelationshipType.Equals("enclosure"))
        {
            Console.WriteLine(link.Uri.ToString());
        }
    }
}

```

Figure 11: SyndicationLink Example

not necessarily an intuitive operation. Discovering enclosure metadata required inspecting collections of `SyndicationLink` objects. To help future podcast researchers who may develop systems on this Microsoft platform, an example of discovering podcast episodes is listed in Figure 11.

3.3.3 Design Pitfalls

The benefit of adopting the Python and .NET dependencies was a reduced development cost in terms of time. These design decisions did however introduce

some unexpected problems that required special handling. For the benefit of future researchers interested in adopting similar dependences, these problems and their mitigations are discussed.

Of the 72,786 podcast feeds processed by instances of PodBot, 99.7% completed inspection without an error condition occurring within the crawlers themselves. It may not seem justified to spend the additional development time to add robust error handling into a research project to handle the failures introduced by only 0.3% of the inputs. However, it turned out that the cost of software failures in the crawlers was very expensive given the long runtimes required to process 6.15715 gigabytes of RSS. On average an instance of PodBot was able to process 1039.5 podcast feeds per hour, for a total runtime of about 70 hours. The crawlers ran unattended; therefore any software failure that caused execution to halt added even more time as the computer would sit idle until the failure was noticed by a human.

96% of the software errors originated from unhandled exceptions being thrown from the SyndicationFeed .NET framework object (see Table 5). Exception handling was added to the PodBot for the purpose of logging each of these exceptions so that a failure analysis could be performed. 4% of the software failures, caused by only 9 individual podcast feeds, forced the Python runtime environment to enter an unrecoverable state, whereas the spawned process required termination. Thus crawlers must maintain a timer that will automatically kill the Feed Validator child process after a timeout period. Log data suggests a timeout period of 10 minutes is sufficient.

3.4 Measurements

Before the data quality of podcasting can be improved, the data in podcasting systems must be measured and analyzed so as to build an understanding of where data

# Exceptions	.NET Framework Exception
85	An error was encountered when parsing a DateTime value in the XML.
36	Text' is an invalid XmlNodeType.
29	An error was encountered when parsing the item's XML. Refer to the inner exception for more details.
21	'Element' is an invalid XmlNodeType.
20	The feed being deserialized has non-contiguous sets of items in it. This is not supported by 'System.ServiceModel.Syndication.Rss20FeedFormatter
5	Unexpected node type Comment. ReadElementString method can only be called on elements with simple or empty content.
5	For security reasons DTD is prohibited in this XML document. To enable DTD processing set the DtdProcessing property on XmlReaderSettings to Parse and pass the settings into XmlReader.Create method.
3	The element with name 'script' and namespace " is not an allowed feed format.
3	The element with name 'br' and namespace " is not an allowed feed format.
2	The remote server returned an error: (503) Server Unavailable.

Table 5: Exceptions thrown from **SyndicationFeed**

Field	Description
feedURL	The URI to the podcast feed hosted on the third party web server.
Popularity	A value between 1 and 0 that provides a relative Popularity rating of the series within a category.

Table 6: The iTunes inputs to PodBot

quality problems exist. The tool described in the previous section was tasked with performing these measurements. The data set that was measured and the findings of PodBot are described here.

3.4.1 The Data Set

The Apple iTunes catalog was chosen as the source of the data set for this project. Exposed to users through the iTunes Store, this online catalog contains a repository of podcast feeds. Each catalog entry (see Figure 12) contains a URI to a podcast feed and a variety of attributes that Apple has associated with the feed. Some of these attributes comes from the podcast feeds itself, such as a text description of what the podcast series is about. It is understandable that Apple would essentially store a duplicate of metadata already contained in the RSS feed in its own database as this allows the iTunes software client to display metadata that describes a particular show without having to make a request against a third party web server. However some of these attributes are specific to iTunes such as an iTunes popularity value and an iTunes category label. It is these iTunes specific attributes listed in Table 6 that serve as the starting point for PodBot.

3.4.2 Categories

The iTunes podcast catalog is divided into 16 subject matter categories. Each of these subject matter categories are further divided into subcategories that relate to the parent in some way. An example of such an iTunes hierarchy is the Arts

```

<dict>
  <key>artistId</key><integer>204040224</integer>
  <key>artistName</key><string>TEDTalks</string>
  <key>buyParams</key><string>productType=podcast</string>
  <key>category</key><string>Design</string>
  <key>description</key><string>TED is a nonprofit devoted to Ideas Worth Spreading.</string>
  <key>episodeGUID</key><string>eng.video.talk.ted.com:1311</string>
  <key>episodeURL</key><string>http://feedproxy.google.com/AlbertoCairo_2011X.mp4</string>
  <key>explicit</key><integer>0</integer>
  <key>feedURL</key><string>http://feeds.feedburner.com/TEDTalks_video</string>
  <key>genre</key><string>Design</string>
  <key>genreId</key><integer>1402</integer>
  <key>itemId</key><integer>160892972</integer>
  <key>itemName</key><string>TEDTalks (video)</string>
  <key>keywords</key><string>TED TEDTalks TED Talks inspiration creativity tech demo education</string>
  <key>kind</key><string>podcast</string>
  <key>longDescription</key><string>TED is a nonprofit devoted to Ideas Worth Spreading.</string>
  <key>playlistId</key><integer>160892972</integer>
  <key>playlistName</key><string>TEDTalks (video)</string>
  <key>podcast-id</key><integer>160892972</integer>
  <key>podcastId</key><integer>160892972</integer>
  <key>podcastName</key><string>TEDTalks (video)</string>
  <key>popularity</key><string>0.5696028</string>
  <key>priceDisplay</key><string>Free</string>
  <key>releaseDate</key><string>2011-12-23T15:02:00Z</string>
  <key>s</key><integer>143441</integer>
  <key>subscribeURL</key><string>https://buy.itunes.apple.com/WebObjects/MZFinance.woa/wa/subscribePodcast?id=160892972</string>
  <key>url</key><string>http://itunes.apple.com/us/podcast/tedtalks-video/id160892972</string>
  <key>is-video</key><true/>
</dict>

```

Figure 12: Example iTunes Catalog Entry

category, which contains the following subcategories: All, Design, Fashion & Beauty, Food, Literature, Performing Arts, and Visual Arts. The All subcategory is the only subcategory common to all 16 parents. It was these All subcategories that were queried for this project. Table 7 lists how many feeds were found in each All subcategory. Together the 16 All subcategories yielded 95,502 entries, each containing a podcast feed. It should be noted that this collection of entries does not represent the entire iTunes podcast catalog, as it was found that there are some entries contained in subcategories that were not included in the All subcategories. Inspecting a subset of subcategories found that this number was small, and that the 95,502 feeds are sufficiently representative of all feeds contained within the iTunes catalog.

Some feeds had more than one entry in the iTunes catalog. Mostly this occurred when a series was included within multiple categories. However it was also the case that a feed occurred within the same category multiples times. Interestingly each of these entries had their own unique popularity value, which indicates that these duplicates are being tracked as separate series from the perspective of iTunes.

iTunes Category	# Feeds
Arts	6000
Business	6000
Comedy	6000
Education	6000
Games & Hobbies	6000
Government & Organizations	5502
Health	6000
Kids & Family	6000
Music	6000
News & Politics	6000
Religion & Spirituality	6000
Science & Medicine	6000
Society & Culture	6000
Sports & Recreation	6000
Technology	6000
TV & Film	6000

Table 7: Number of feeds in each iTunes All Subcategories

We consider this a defect of the iTunes catalog. Therefore PodBot removed 22,716 duplicate feeds from the sample set, so as to not skew the measurements. The table below contains the number of duplicates discovered as the PodBot processed the raw iTunes XML. The iTunes categories were processed in alphabetical order, therefore the number of duplicates generally increased from Arts to TV & Film. Finally, after having ingested the iTunes XML and removed duplicate entries, we discovered 72,786 unique feeds.

3.4.3 Popularity

Previous research into podcasting has used iTunes popularity as a dimension in which to compare various podcasting attributes [18, 42]. These previous efforts were somewhat simplistic in so far as they simply observed the ordered list of shows displayed through the iTunes user interface. The implication is that the researchers had no concept of the degree to which one show is more popular than another. An

iTunes Category	Duplicates Discovered	Feeds Ingested
Arts	27	5973
Business	181	5819
Comedy	486	5514
Education	1200	4800
Games & Hobbies	1198	4802
Government & Organizations	446	5056
Health	1098	4902
Kids & Family	1274	4726
Music	1210	4790
News & Politics	1757	4243
Religion & Spirituality	1218	4782
Science & Medicine	1742	4258
Society & Culture	3664	2336
Sports & Recreation	1398	4602
Technology	2844	3156
TV & Film	2973	3027

Table 8: Duplicate feeds discovered in iTunes

ordered list does not properly convey the change in the rate of popularity. This thesis inspects the actual iTunes popularity value captured by making queries against the iTunes cloud service. Given that there is interest in the research community in gaining an understanding of iTunes popularity, a discussion of these values follows.

Each entry in the iTunes podcast catalog contains a popularity field. The ranges of popularity values are specific to a category. It was observed that each iTunes podcast category has its own range of popularity values. This range was a real number between 1 and a small number approaching, but not equal to, zero (the smallest popularity value was 0.0000016745482). It was found that individual shows could be listed within the catalog multiples times. Multiple entries occurred when a show was listed under different categories. For example, the show *The Moth Podcast* (see Table 9) was listed under both the Comedy and the Arts categories. In these cases, each entry had its own unique popularity value. That is, a single show could have multiple popularity values that are specific to a category context. It can be

Feed	iTunes Category	Popularity
http://feeds.feedburner.com/themothpodcast	Arts	0.2301070800
http://feeds.feedburner.com/themothpodcast	Comedy	0.7939343000

Table 9: Example of a Duplicate feed

assumed then that a popularity value from one category does not have meaning when compared to the popularity values of other categories.

Although Apple does not publish how the popularity values are calculated, we can make a few observations through manual inspection. The show listed within the iTunes user interface as being the most popular show per category has a popularity value of 1. Furthermore, each category has one and only one show with a popularity value of 1. Therefore, we can surmise that 1 indicates the show is the most popular entry for that category. The rest of the shows within a category have values less than 1 but greater than 0. It is assumed that this represents the distance a given show is from the most popular for that category. That is, it is assumed a show with a popularity value of .5 is half as popular as the most popular show and that a show with a value of .25 is only one fourth as popular as the top show. Unfortunately this data does not provide guidance towards the actual number of subscribers per show. But if we could learn the number of users that have subscribed to a show with value of 1, we could then make estimations of subscribership by multiplying the popularity values by the number of subscribers.

3.4.4 Metrics

As a team member at Microsoft during the development of Zune, it was observed that podcast feeds that failed RSS 2.0 validation were the leading cause of user experience failures occurring within the podcast functionality of the Zune software. From the user perspective these failures often manifested themselves as feeds that

Feed	Popularity
http://feeds.thisamericanlife.org/talpodcast	1.0000000000
http://feeds.feedburner.com/comedycentral/standup	0.2909667500
http://feeds.feedburner.com/themothpodcast	0.2301070800
http://americanpublicmedia.publicradio.org/podcasts/xml/prairie_home_companion/news_from_lake_wobegon.xml	0.1780055500
http://www.qdnow.com/grammar.xml	0.1394481800
http://feeds.feedburner.com/TEDTalks_video	0.1343486000
http://selectedshortspri.pri.libsynpro.com/rss	0.1080078700
http://wtfpod.libsyn.com/rss	0.1031321360
http://feeds.newyorker.com/services/rss/feeds/fiction_podcast.xml	0.0986269040
http://www.gcast.com/u/dane_cook/main.xml	0.0960611200

Table 10: Top 10 Most Popular Feeds in Arts

would simply not display within the client. Given that users do not often understand the source of failures, it reflected poorly upon the client software itself rather than the author of the invalid feed. However, even podcast feeds that passed validation were found to cause failures.

To build an understanding of how dire the problem of invalid podcast feeds is to the health of the podcasting ecosystems, the PodBot performed a validation against every feed visited, the results are listed in Table 11. It was found that 30,580 or 42.01% of all podcast feeds in the data set failed RSS 2.0 validation. This is an alarmingly large percentage. The crawlers encountered an HTTP request failure rate of 13.28%. That is, some feeds were unreachable, and therefore a validation attempt could not be made. Removing the unreachable feeds from the failure calculations, the failure rate climbs to 48.63%. Table 12 and 13 list the number of valid and invalid feeds discovered in each iTunes category. Table 14 compares the most popular podcasts of each category against the least popular podcasts of each category.

Previous research projects have attempted to build a model of podcasting [4, 18]. This project contributes to this body of work. An accurate model of podcasting will be particularly interesting from a software quality assurance perspective. Soft-

RSS 2.0 Validation	# Feeds	Feeds %
Valid	32308	44.39%
Invalid	30580	42.01%
HTTP Failure	9669	13.28%

Table 11: Validation Results for All podcasts

iTunes Category	# Valid	% Valid
Society & Culture	3083	51.3833%
Art	2991	49.8500%
Health	2901	48.3500%
Kids & Family	2896	48.2667%
Games & Hobbies	2895	48.2500%
Comedy	2854	47.5667%
Religion & Spirituality	2809	46.8167%
Technology	2797	46.6167%
TV & Film	2772	46.2000%
Music	2739	45.6500%
Science & Medicine	2721	45.3500%
Education	2625	43.7500%
Sports & Recreation	2582	43.0333%
Government & Organization	2252	40.9306%
Business	2396	39.9333%
News & Politics	2379	39.6500%

Table 12: Valid Feeds by Category

iTunes Category	# Invalid	% Invalid
Kids & Family	2098	34.9667%
Society & Culture	2276	37.9333%
Health	2285	38.0833%
Music	2342	39.0333%
Art	2369	39.4833%
Games & Hobbies	2373	39.5500%
Comedy	2392	39.8667%
TV & Film	2403	40.0500%
Sports & Recreation	2474	41.2333%
Science & Medicine	2485	41.4167%
Technology	2499	41.6500%
Education	2504	41.7333%
Government & Organization	2328	42.3119%
Religion & Spirituality	2550	42.5000%
Business	2747	45.7833%
News & Politics	2866	47.7667%

Table 13: Invalid Feeds by Category

iTunes Category	# %Valid Top 50	% Valid Bottom 50
Art	44%	46%
Business	42%	42%
Comedy	58%	58%
Education	48%	46%
Games & Hobbies	32%	60%
Government & Organization	40%	42%
Health	72%	62%
Kids & Family	52%	52%
Music	40%	34%
News & Politics	30%	34%
Religion & Spirituality	62%	52%
Science & Medicine	42%	46%
Society & Culture	44%	54%
Sports & Recreation	28%	46%
Technology	50%	46%
TV & Film	42%	58%

Table 14: Invalid Feeds by Popularity

ware testing professionals often employ Boundary-Value analysis [36]. Given that testing all possible inputs into a system is often not possible for any sufficiently complex system, Boundary- Value analysis attempts to classify inputs into related sets with the upper and lower bounds of the sets being values targeted for testing. The data provided here will be valuable to software testing professionals responsible for ensuring the quality of podcasting systems, as these measurements can be used to identify boundary values.

Tables 15 and 16 list the size of the RSS feeds and various elements contained within the feeds. We found the median valid podcast feed was around 40 KB, but invalid podcast feeds came in around 25 KB. More investigation will be needed to understand this size discrepancy. Tables 17 and 18 list the length and size of podcast episodes. The median length of an episode is slightly longer than 33 minutes, and comes in at a size of almost 17 MB. Table 19 shows that the median number of episodes per series is 18. Therefore, we can estimate that an average feed hosts 306 MB worth of content. Tables 20 and 21 list the most popular media formats for a podcast episode (see Table 53 in the appendix for a complete listing of media formats found by PodBot). We found over 77% of all podcast episodes are audio. Table 22 lists the most popular XML namespace extensions used within podcast feeds. Not surprisingly the iTunes namespace was found in 100% of the sample set. This is understandable given that Apple requires this extension as a prerequisite for a feed being included within its catalog (see Table 55 in the appendix for a complete listing of namespaces found by PodBot). Table 23 lists of language of the podcast content as specified by the human podcaster. Here we find that over 90% of the feeds are some variant of English (see Table 56 in the appendix for a complete list of language codes found by PodBot). Finally Table 24 lists the values found in the generator field. Unfortunately the most popular value was the empty field at 25%. This suggests that

Feed	Mean	Median	Min	Max
Valid	110710.85	39838	234	8624999
Invalid	99700.59	24750	0	7190635

Table 15: Size of podcast Feeds in Bytes

Element	Mean	Median	Min	Max
title	27.81	24	0	315
description	216.87	123	0	6317
item title	39.46	35	0	1449
item summary	685.45	273	0	555990
item URI	72.25	71	0	504

Table 16: Size of RSS Elements in Characters

there are RSS authoring tools that do not identify themselves, or perhaps these are feeds that were hand authored by the podcaster (see Table 57 in the appendix for a complete listing of generators found by PodBot).

We have now discussed the measurement phase of our TDQM process. The RSS 2.0 validation service hosted by the W3C and the open source Feed Validator project were described. We provided an overview of the architecture and lessons learned from the development of PodBot. PodBot discovered that 42% of all podcasts founds within the iTunes catalog fail RSS 2.0 validation. And finally PodBot built a model of an average podcast feed and episode.

Mean	Median	Min	Max
00:40:21.68	00:33:38:00	0:00:00	23:48:54

Table 17: Length of podcast Episodes (hh:mm:ss)

Mean	Median	Min	Max
90473163.66	16967345.00	0	9223372036854775807

Table 18: Size of podcast Episodes (Bytes)

Mean	Median	Min	Max
42.85	18.00	0	3828

Table 19: Number of Episodes per podcast Series

MIME Type	Episode %
audio/mpeg	77.10%
video/mp4	5.54%
video/x-m4v	4.31%
EMPTY FIELD	4.16%
audio/x-m4a	2.66%
video/quicktime	1.43%

Table 20: Top 5 Episode Format Types

MIME Type	Mean	Median	Min	Max
audio/mpeg	98121994.54483	17387947	0	9.22337E+18
video/mp4	142006132.98446	36610336	0	2.415E+11
video/x-m4v	80987019.15629	30209162	0	16388000000
audio/x-m4a	33270999.27017	19772035	0	32108669329
video/quicktime	71296570.38464	26035364	0	4294967295

Table 21: Episode Sizes by Format

Namespace	Feeds %
http://www.itunes.com/dtds/podcast-1.0.dtd	100.000%
http://search.yahoo.com/mrss/	43.917%
http://purl.org/dc/elements/1.1/	37.027%
http://rsshnamespace.org/feedburner/ext/1.0	32.931%
http://purl.org/rss/1.0/modules/content/	27.861%

Table 22: Top 5 XML Namespaces

Language	Feeds %
en	58.500%
en-us	31.945%
en-gb	1.993%
EMPTY FIELD	1.396%
de	1.192%

Table 23: Top 5 Languages

Language	Feeds %
EMPTY FIELD	25.642%
Libsyn WebEngine	7.993%
http://podbean.com/?v=3.2	5.484%
podOmatic RSS Generator	4.852%
http://wordpress.org/?v=3.3.2	4.266%

Table 24: Top 5 RSS Authoring Tools

CHAPTER IV

ANALYZE PODCASTING

During the analysis phase of the TDQM methodology quality professionals will evaluate actual data quality failures in the information manufacturing system under inspection. Through this investigation the root cause of data quality failures will emerge, and specific problems can be identified. It is only after the specific instances of data quality problems are identified and understood, that specific actions for improving data quality can be planned.

4.1 Quality Problems

Violations of the data quality requirements for podcasting are now examined. How each violation causes failures in popular podcasting systems is demonstrated and discussed. Through analysis of these data quality violations and the quantitative measurements, an understanding of how podcasting data can be improved will be built.

4.1.1 Correctness Problem

The most fundamental of the data quality requirements for podcasting, and perhaps for any data manufacturing system, is the factual correctness and structure of the data itself. This thesis focuses only on the metadata that describes a podcast series and podcast episodes that are contained within a podcast feed. Metadata that describes podcast episodes can also be found within the multimedia files themselves. In the context of audio, metadata can be stored within the ID3 fields of an MP3 file. However, neither this project, nor previous podcast research [4, 18] have investigated the metadata contained within multimedia files. We decided not to investigate due to the size difference and thus the time necessary to capture multimedia. That is, the mean podcast feed was found to be 110 KB whereas the mean podcast multimedia file was found to be 90 MB. Furthermore, no authoritative organization has published recommendations for podcasting metadata stored within media files. Thus it would be difficult deciding which fields to inspect.

Internet data must be structured in a manner that is well understood so that independent software systems are capable of processing content described by the metadata. In the context of podcasting this well understood structure is the RSS 2.0 standard. Failure to conform to this standard is an example of violating the correctness property of podcasting. It was observed during the measurement phase that 42% of all podcast feeds violate at least one rule of the RSS 2.0 specification. Each of these violations was captured and yielded 110,614 validation errors with 15,998 unique error messages. Table 25 lists the most popular RSS 2.0 violations. A complete listing of error messages generated from the set of iTunes feeds is listed in Table 34 of the appendix.

Correctness can be violated even within valid RSS feeds. The values of the various RSS fields could contain factually incorrect information. Some factual viola-

# Feeds	Error
2405	Undefined root element: xhtml:html
2084	Undefined root element: script
2024	XML parsing error: not well-formed (invalid token)
2012	link must be a full and valid URL
1897	Invalid email address
1480	Incorrect day of week
1394	pubDate must be an RFC-822 date-time
1254	Invalid character in a URI
1184	Undefined channel element: itunes:link
1097	Undefined root element: html

Table 25: The Top 10 RSS 2.0 Errors

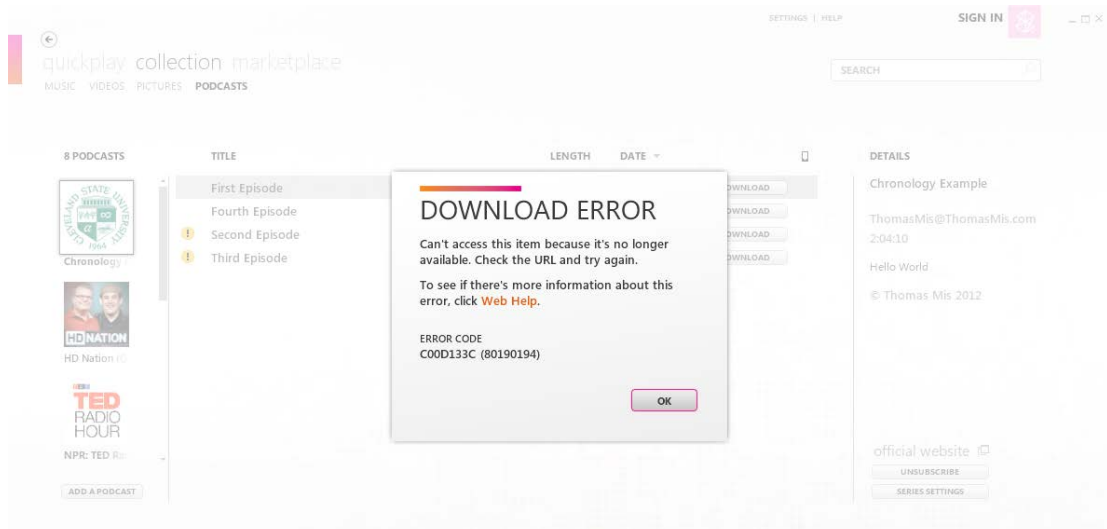


Figure 13: Example of Unresolved URI in Zune

tions may not necessarily cause serious user experience failures in podcasting systems. For example, the description field for a newly released podcast episode may mistakenly contain the description for a different previously released episode. In this case the user may read this incorrect description value, and choose to not listen to the episode. This is certainly a data failure, but it does not cause either iTunes or Zune to improperly function.

A more severe example of factual violation is a feed that contains incorrect episode size or duration information. Software clients may make decisions based on

the metadata contained within the feed. If software on a mobile phone attempts to avoid downloading large podcast episodes, then the correctness of the size metadata is important. A feed that contains a size value smaller than the actual size of the media file could cause the mobile phone to exceed the bandwidth limitation imposed by the mobile service provider. In the next chapter, we show that our improved validator does indeed check this metadata field, and found that 44.5% of feeds contain incorrect size data.

The most severe failure is a feed that contains an incorrect URI to a podcast episode. Again in the next chapter we show that the improved validator attempts to acquire episode in order to validate the URI values. It was found that 18.66% feeds contained episodes that could not be downloaded. Figure 13 illustrates how an incorrect URI is displayed within the Zune user interface.

4.1.2 Uniqueness Problem

Violations of the uniqueness data quality requirement cause podcasting systems to fail at determining whether an episode of a podcast series is unique. The RSS specification calls for a globally unique identifier (GUID) to be associated with each podcast episode. This GUID field should be sufficient for helping podcasting systems to distinguish between different episodes in a series. Unfortunately the existing feed validator does not enforce any standards around the values that are used as identifiers. Therefore, it was found during the measurement phase that all manner of string data were being used as globally unique identifiers.

Of the 1,091,295 episode GUID values programmatically inspected, it was found that 766,292 or 70.22% of the GUID values began with the substring **http://**. We found that in these cases the GUID value generally matched the corresponding enclosure value for the inspected podcast episode. Table 26 provides samples of the

GUID Example
5577 at http://www.thisamericanlife.org http://download.publicradio.org/podcast/nflw/2012/04/28/nflw_20120428_64.mp3 77f68e0396964a0182915c0b99219f41 tag:blogger.com:1999:blog-33028507.post-7703483890054156235 0455a25f-2398-429b-8961-7a0ad5f1eb73 4088 77D1E331-B835-4DDF-A81F-4A4CCEE3CE0B-233-00000C953049E486-FFA sonibyte-18637.mp3 vineyard-development-napa-valley-wine-radio S http://cni.libsyn.com/index.php?post_id=96127# .CZiKZiuM5Q /?p=63 ??-????-????-?????-????? 00AA6082-2D5A-4ADB-B17F-84E01CC56FA6 021-10.25.09 hiphop-podcast-18

Table 26: Examples of GUIDs Discovered by PodBot

kinds of GUID values found during the measurement phase.

There is nothing that prevents podcasters from recycling these GUID values as old episodes are removed, and new episodes are added to podcast feeds. Though inspection, it was found that many podcast series only keep a subset of episodes listed within a feed. Most often this is the most current set of episodes, such that the number of episodes found within the feed is constant. In this example, the feed can be described as a FIFO queue, where the episode being removed is the oldest episode contained in the feed. The uniqueness data quality property is violated if the GUID from the episode being removed is used for the episode being added. It can be speculated that many podcasters are not computer scientists, and therefore do not understand the purpose of the GUID field.

Given these reasons, the Zune software does not use the GUID field to determine if a podcast episode is unique. Rather, the title and episode URI together are considered sufficient to determine uniqueness. This heuristic assumes that the

```

<item>
  <title>Example Episode 1</title>
  <link>http://www.podtrac.com/twit0359.mp3</link>
  <description>Hello World</description>
  <author>ThomasMis@ThomasMis.com</author>
  <pubDate>Sun, 24 Jun 2012 18:08:10 -0700</pubDate>
  <category>News</category>
  <category>Technology</category>
  <comments>http://twit.tv/show/this-week-in-tech/359</comments>
  <enclosure url="http://www.ThomasMis.com/twit0359.mp3" length="59729675" type="audio/mpeg"/>
  <guid isPermaLink="false">A3AB9D77-CDB1-4DA5-88D5-9466A893BF65</guid>
</item>

```

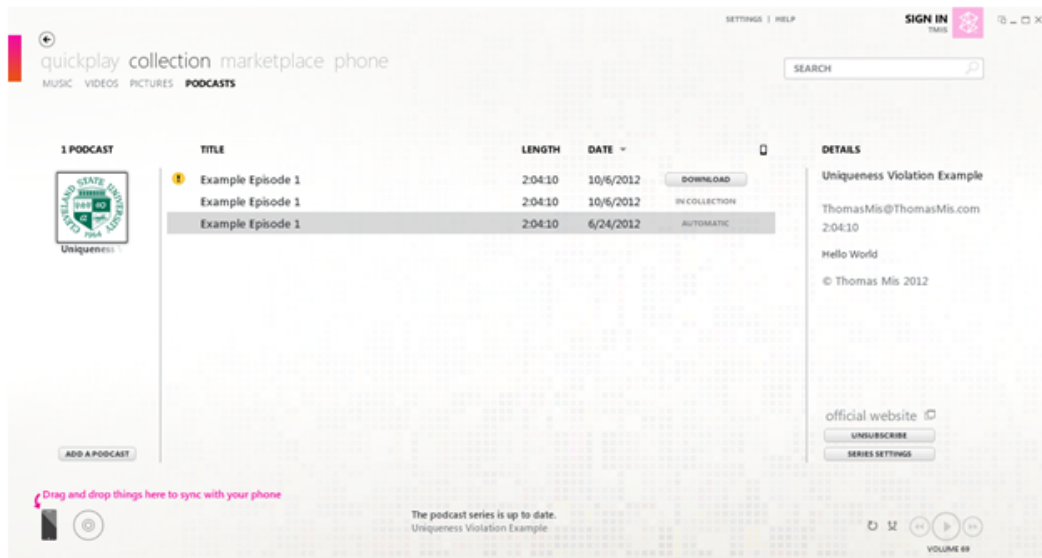


Figure 14: Example of Uniqueness Bug in Zune

title and URI will not be changed after the feed is first aggregated by Zune. Software failures thus manifest in Zune when podcasters find the need to edit an already published episode. It was observed during maintenance of the Zune client that these edits occurred most often when episodes were moved from one hosting provider to another. The impact in software clients was the introduction of false episodes listed in the client UI. Figure 14 illustrates the impact of changing the URL attribute of the enclosure element from podtrac.com to thomasmis.com. Editing the XML in this manner cause episodes to appear in triplicate within Zune and duplicate within iTunes.

4.1.3 Platform Adherence Problems

The commercial podcast platforms publish guidelines that podcasters should adhere to when producing digital media. These guidelines generally cover aspects of media production that are not defined by the formal rules of RSS. These supplemental guidelines are provided to ensure that the media produced by podcasters is compatible with the software and devices sold by the commercial vendors. Examples of the rules provided by Microsoft and Apple are listed in Table 2 and Table 1. Unfortunately the podcasting platforms from these vendors support divergent sets of media formats. An episode encoded to the QuickTime specification may playback on an iPhone device, but it may not playback on a Windows Phone device. Podcasters should use these guidelines to choose media formats that support the widest possible set of platforms.

From the data collected during the measurement phase, we observed podcast feeds that support non-compliant media types. Of the 30 most common media formats discovered during the measurement phase (see Table 54), we found 6 type that are not supported on either iPod or Zune devices. Table 27 lists these unsupported formats. Figure 15 illustrates the Zune the user experiences when attempting to interact with

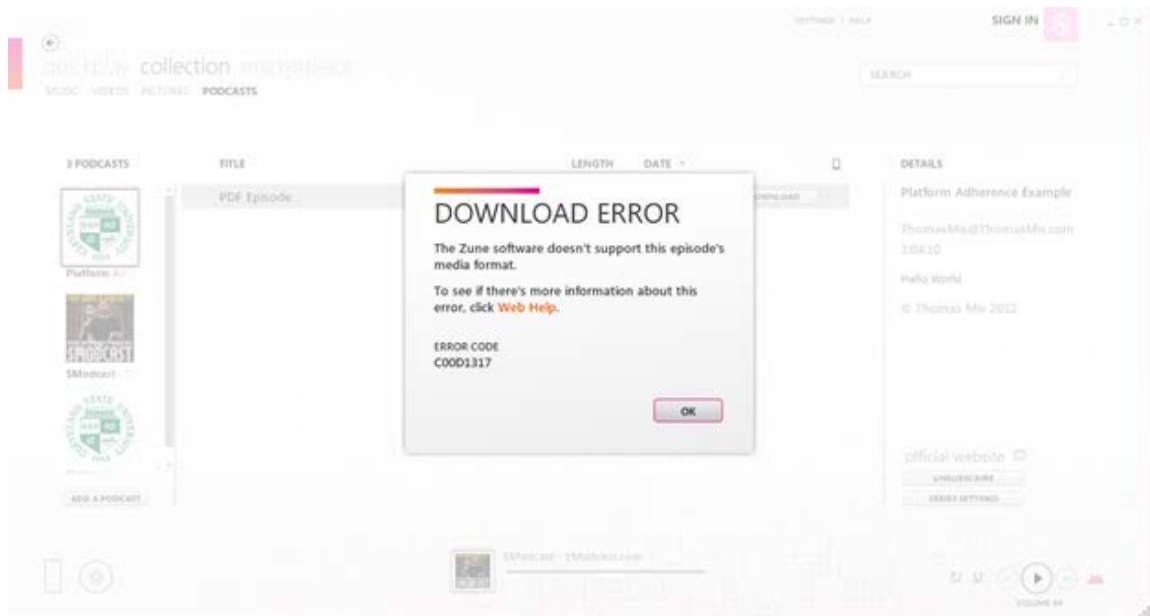


Figure 15: Example of Platform Adherence Bug in Zune

Rank	Format
9	application/pdf
14	application/octet-stream
20	application/x-shockwave-flash
22	test/plain
25	video/x-flv
28	text/html

Table 27: Top Unsupported Formats Found by PodBot

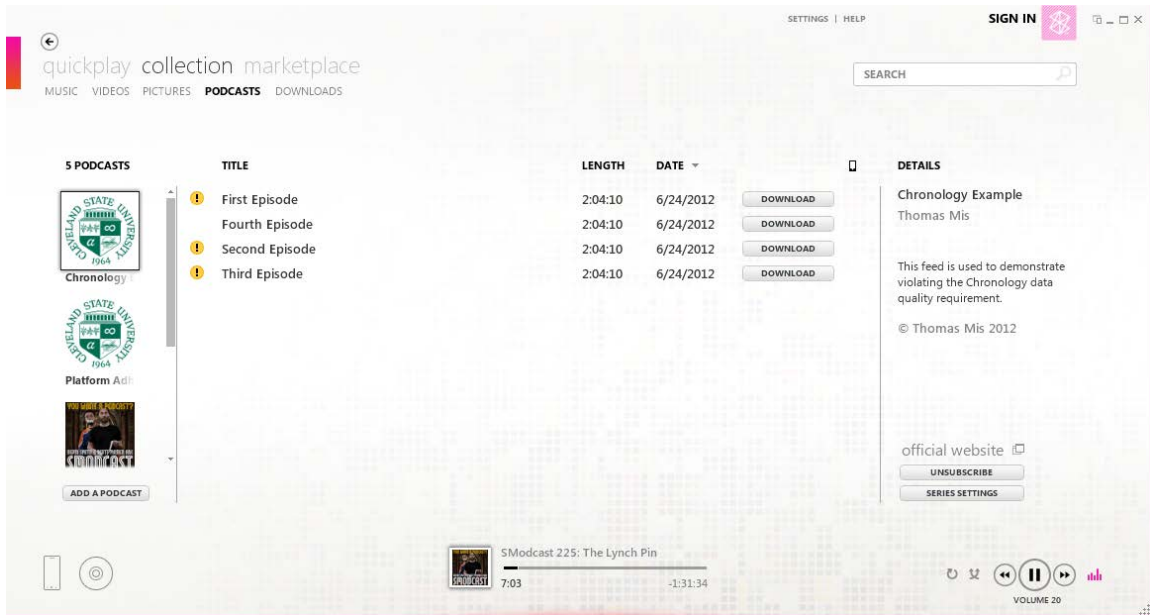


Figure 16: Example of Chronology Problem in Zune

an unsupported podcast episode.

4.1.4 Chronology Problems

Podcast episodes within a series have a chronological ordering. Podcast aggregators respect this ordering by listing episodes based on the item publication date. This publication date is specified as part of the RSS standard. The podcast data quality property of chronology therefore states that podcasting systems must be able to determine the proper chronological ordering of a podcast series. Although the RSS validator does check for the presence of a publication date, it does not validate the publication date is unique or ordered properly within a feed. An example of violating the chronological principle occurs when podcasters release multiple episodes simultaneously.

In the case of simultaneously published episodes, it was found that podcast aggregators are unable to determine chronological ordering, and therefore the ordering of episodes within the UI was not deterministic. This may be an inconsequential

behavior if the episodes within a podcast series contain content that is independent of other content. This behavior becomes a software failure if the content is dependent on ordering. This issue could be avoided if the podcasters simply increment the minute or seconds value of the publication date for each episode. Figure 16 shows four episodes published on the same feed, each with the exact same publication date. Notice that the episodes are listed in alphabetical ordering, which in this case has caused the fourth episode to be displayed second.

4.1.5 Performance Problems

The RSS specification does not define constraints around file sizes or episode limits. Therefore violating the performance data quality principle can be platform dependent. An example of such a platform dependent violation could be the publishing of a multimedia file that is too large to be downloaded to a mobile phone that has a data download limit. In this case the platform is the mobile telecommunications network. A more specific example of a performance violation is a podcast feed that is larger than 11 MB, as this will violate the file size limit imposed by the Zune software client (in this case the client simply refuses to attempt to parse the feed). To circumnavigate such performance failures, podcast publishers have released multiple feeds for the same podcast series where each feed has different performance characteristics.

The American mobile telephone service provider AT&T currently limits its customers to 3GB of data transfer per month. The show *This Week in Tech* is a weekly podcast that provides video files on the order of 300 MB per episode. Therefore, an AT&T customer who subscribes to this podcast will transfer 1.5 GB of data per month for this series alone. If the user subscribes to a second show that contains episodes of similar size, the data limit will be reached each month. A third show would cause the data limit to be exceeded. We consider this to be a violation of the

Performance data quality requirement in the context of mobile telephones.

4.2 Analysis Conclusions

Based on analyzing examples of real world violations of the podcast data quality principles, it can be stated that valid RSS 2.0 podcast feeds can cause data quality failures in popular podcasting software systems. Therefore RSS validation alone is insufficient to prove that a podcast feed is free of data defects.

CHAPTER V

IMPROVE PODCASTING

The TDQM methodology provides a framework around which an organization can improve the quality of data. The measurement and analysis phases of the methodology serve to help information quality professionals understand data quality problems, and identify areas for improvement. In this thesis, we have conducted an extensive measurement of over seventy thousand podcasts, and have analyzed failures in popular commercial podcasting systems caused by defects in podcast data. From this measurement and analysis we conclude that the existing data validation service provided by the World Wide Consortium is insufficient at ensuring an RSS 2.0 compliant podcast feed is free of defects that cause failures. Thus, information quality in podcasting will be increased and failures in podcasting systems will be reduced through the use of an improved validation service. This chapter describes the construction and use of this improved validation service.

5.1 PodCop Overview

The culmination of the measurement and analysis activities heretofore described is the creation of an improved podcast validation service — *PodCop*. This improved podcast validator understands the information quality requirements that are unique to syndicating podcast episodes, whereas the existing validator hosted by the W3C does not make a distinction between podcast syndication and plain text syndication. A discussion of PodCop follows.

PodCop was created for this thesis in order to demonstrate how extending the RSS specification with a small number of podcast specific rules can greatly impact the quality of podcasting feeds. A sampling of RSS 2.0 compliant podcast feeds from the measurement phase of the project were evaluated with the PodCop validator. It was found that 66.5% of the valid feeds contained violations of the extended rules. To encourage software engineers and academic researchers to further explore and improve podcast validators, an overview of the PodCop architecture is provided.

The architecture of the improved validator is similar to that of the PodBot crawler created for the measurement phase. PodCop was built upon the same code-base and leverages the same SyndicationFeed Microsoft .NET class. The main differentiator between the two projects is that in PodCop, the open source Feed Validator component is replaced with a custom component that encapsulates the extended podcast rules. This new component contains six simple rules that provide validation for some of the podcast information quality requirements defined in Chapter 2. This component is a proof of concept, and is intended to motivate the creation of a more extensive podcast validator for podcast producers.

Although the architectures of the two systems created for this thesis are similar, the performance characteristics of each of the systems are very different. A goal of the measurement phase of was to capture metrics from the widest possible sample set

Set	# Feeds	Runtime (Hours: Minutes)
Random Set - A	100	18:08
Random Set - B	100	48:42
Random Set - C	100	21:40
Random Set - D	100	14:25
Random Set - E	1000	229:57
Random Set - F	1000	239:06
Popular Set - G	194	38:43

Table 28: PodCop Runtimes

of feeds. It was therefore decided that only the feed itself, and not the multimedia files referenced from those feeds would be downloaded and inspected. The benefit of this design decision was that the total runtime of the PodBot was around 70 hours, or 3.46 seconds per podcast feed. However, in order to enforce some of the quality requirements defined for podcasting data (such as the correctness requirement), the PodCop validator must actually download each of the episodes in a podcast series. This causes considerable performance degradation when compared to the crawler. PodCop was only able to perform at the rate of 880.82 seconds per podcast feed (14.68 minutes per feed). Given this long runtime PodCop was executed upon a subset of the feeds from the measurement phase. We examined both a random subset of RSS 2.0 compliant feeds and the set of most popular RSS 2.0 compliant feeds from each iTunes category. Table 28 lists the execution time of PodCop for each set.

5.1.1 Enforcing Correctness

This information quality principles states that the metadata contained within a podcast feed should be factually correct. Some fields of a podcast feed are difficult to programmatically evaluate. An example of a difficult to evaluate field is the contact information for the human responsible for maintaining a podcast feed. A system could be devised such that the validator software attempts to contact the human

podcast producers through email, and withhold declaring a feed to be valid until the human responds. However, even in this scenario, the software could not be absolutely certain the email response still came for the correct human if the feed has been compromised and the contact information falsified by a malicious person. Furthermore, a design goal of PodCop is to have relatively similar performance to the open source Feed Validator project which operates on RSS feeds on the order of seconds. Therefore, validations that require human interaction are too slow for the purposes of this project.

From the perspective of reducing software failures, a podcast feed containing an incorrect email address may be deemed acceptable if the email address does not cause unexpected operation of the podcasting client. The same can be said for the text description field of an episode. If the podcast producer uploads text data that contains grammatical errors, or even if the text describes the wrong show, the podcast client will operate correctly and display the incorrect text. For this project, these kinds of correctness failures are considered acceptable.

The PodCop validator attempts to identify three possible defects within podcasting feeds that can cause operational failures in podcasting software: URI to the series image, URI to individual media files, and the size attribute of each media file. Validating the URI involved verifying that the remote host returned the requested file. Validating the size attribute involved comparing the value provided by the podcast producer with the actual file size of the requested media file. File size failure was called out by [18] as a common defect with podcasting metadata, with 35% of the file size attributes containing incorrect values. PodCop confirms this high failure rate.

5.1.2 Enforcing Uniqueness

The uniqueness information quality principle is violated when the provided metadata is insufficient in deterring that a podcast episode is unique. As mentioned earlier the root cause of uniqueness defects comes from podcast client software that does not trust the GUID values provided by feed authors. PodCop enforces the uniqueness principle by checking that each episode URI and episode title fields are unique to a series. Merging the URI and title field is a substitute for referencing the GUID value in determining uniqueness in both of the popular podcast aggregators. Therefore, it is the combination of these fields that must be deemed unique. However, this project enforces a stricter rule that both the URI and title fields must be unique independently.

5.1.3 Enforcing Chronology

The chronology podcast quality principle is violated when the publication dates for Podcast episodes do not reflect the intended ordering of episodes within an aggregator. As demonstrated in the previous chapter, this principle is often violated when podcasters releases multiple episodes with the same publication date values. Therefore, the PodCop validator checks the publication date element of each episode for uniqueness.

5.2 PodCop Results

2,594 podcast feeds were evaluated with this improved validator. Only podcast feeds that had passed RSS 2.0 validation with the W3C validation tool were considered as candidates for inspection with PodCop. The measurement phase identified 32,308 feeds (44.39% of the iTunes catalog) that conformed to the RSS 2.0 specification.

Requirement	Rule	Failure %
Correctness	Episode URI does Resolve	18.66%
Correctness	Image URI does Resolve	7.48%
Correctness	Episode Size is Correct	44.56%
Uniqueness	Episode URI is Unique	15.84%
Uniqueness	Episode Title is Unique	10.61%
Chronology	Episode Publish Date is Unique	10.61%

Table 29: PodCop Failure Rate for Individual Rules

Defect Type	# Feeds	Feed %
RSS 2.0	30580	42.15%
PodCop	21339	29.41%
HTTP	9669	13.33%
No Defects	10969	15.12%

Table 30: Overall Failure Rate for All Podcasts

To reduce the PodCop execution time, a random set of feeds was generated. From the RSS 2.0 compliant set 2,400 feeds were randomly selected. Furthermore, the most popular 15 valid podcasts from each iTunes category were also selected. Some podcasts are popular in more than one category. Removing the duplicate popular feeds yielded 194 additional RSS 2.0 compliant podcasts for inspection. Together the selected set of random and popular feeds represent 8% of the valid feeds and 3.6% of all feeds captured from iTunes.

Overall 66.50% of the RSS 2.0 compliant podcast feeds evaluated with the improved validator failed at least one of the quality requirements. Table 29 breaks the failures down by individual rule. Given that the measurement phase discovered a large number of feeds from the iTunes catalog could not be requested (13.33%) or failed the W3C validation (42.15%), we can predict that 84.88% of all podcast feeds from the iTunes catalog contain defects. Extrapolating the findings from the improved validator over the set of iTunes feeds suggests the following distribution of data defects listed in Table 30.

5.2.1 Random Podcasts

The set of random podcast feeds were evaluated in six separate batches (four lots of 100 feeds and two lots of 1000 feeds). These batches were processed across different computers and separate networks. No significant differences were detected in the results based on computer or network used. Therefore we conclude that these findings can be reproduced independent of the computers and networks used during this thesis. Overall PodCop discovered defects in 66.07% of the randomly selected feeds. The improved validator results for each lot of random feeds are reported in Table 31.

5.2.2 Popular Podcasts

Surprisingly the set of popular podcasts had a higher failure rate when compared to the set of random podcasts. The set of popular podcasts had a failure rate of 71.58%. Table breaks the results down by individual rule. The most noticeable quality difference between the two sets is in the higher occurrence of duplication of episode titles and publication dates. A human inspection of the popular feeds revealed that popular shows from American public radio networks publish multiple media files on the same day with the same timestamps. Although these popular feeds contained more duplication, the media files references from the popular feeds could be retrieved with a higher success rate. This is somewhat understandable as we can expect the popular shows to have more motivation for ensuring their episodes can be reliably retrieved.

In this chapter we have created an improved feed validation service. We have demonstrated that a small number of podcast specific rules can detect the defects that are known to cause quality failures in commercial podcasting systems. Based on the findings of this improved validation service, we predict that nearly 85% of

Requirement	Rule	(A)	(B)	(C)	(D)	(E)	(F)
Correctness	Episode URI does Re-solve	19.57%	22.92%	20.43%	12.63%	18.54%	19.91%
Correctness	Image URI does Resolve	10.61%	8.70%	7.46%	7.46%	7.46%	7.46%
Correctness	Episode Size is Correct	45.00%	45.24%	25.35%	21.05%	46.00%	42.5%
Uniqueness	Episode URI is Unique	10.87%	13.54%	15.05%	21.05%	16.68%	16.79%
Uniqueness	Episode Title is Unique	7.61%	15.63%	16.13%	6.32%	8.83%	8.83%
Chronology	Episode Publish Date is Unique	4.35%	11.46%	11.83%	11.58%	10.14%	10.98%

Table 31: PodCop Results from Random Podcast Feed Sets

Requirement	Rule	Failure %
Correctness	Episode URI does Resolve	12.63%
Correctness	Image URI does Resolve	7.46%
Correctness	Episode Size is Correct	55.91%
Uniqueness	Episode URI is Unique	8.42%
Uniqueness	Episode Title is Unique	20.00%
Chronology	Episode Publish Date is Unique	18.95%

Table 32: PodCop Results from Popular Podcast Feed Set

podcast feeds contain some defect. The use of PodCop can help podcasters prevent a poor user experience for their subscribers.

CHAPTER VI

RELATED WORK

From computer networking researchers to commercial media rating organizations, a variety of research communities have published works exploring the various aspects of podcasting usage and production. We now review these works and where appropriate discuss how this thesis contributes to and extends this body of work.

6.1 Podcasting in Education

The research community has shown a great deal of interest in incorporating podcasting into higher education. This interest is expressed through the publication of papers that examine the applicability of podcasting to the classroom and university administration. Case studies of supplementing and replacing traditional lectures with podcasting are shared, as well as lessons learned from the recording and publishing of podcasts in an academic context.

To understand the attitudes of engineering and computer science students towards podcasting as an education tool, surveys were conducted at a variety of univer-

sities [6, 21, 27, 35]. Students generally viewed podcasting favorably as a supplement to lectures. [21] inverts a semester of his senior software engineering course, where lecturers are given outside of class time in the form of a podcast, and class time is used for projects. The author found that the exam scores of students in the traditional section of the class, and the inverted class were the same. However, the students in the inverted class had project scores that were 10% higher than the traditional class.

Many guidelines have been published that outline recommendations for recording podcast episodes [13, 23, 45, 48] and suggestions for creating engaging academic content for distribution through podcasting. [27][26] attempts to automate the recording process, and describe systems for automatically capturing and publishing academic lectures.

6.2 Podcasting in Mobile Networks

[24] describes a protocol for distributing podcast episodes through a wireless ad-hoc network (such as mobile phones). The protocol calls for each node of the network to maintain two caches of podcast episodes. One cache stores episodes for shows the user has explicitly subscribed to (they call it the private cache), and the second cache is for storing episodes for the purpose of redistributing them to other nodes that may be interested in them (they call it the public cache). The protocol has two phases, during the first phase each node simply asks the other if they have any new episodes for shows that the user has a subscription for (from either cache). The second phase is for populating the private cache in a manner that is healthy for the network overall. The authors investigate which technique is most optimal to ensuring nodes always receive episodes they want in a timely manner. They investigate 5 techniques: Most Solicited, Least Solicited, Uniform, Inverse Proportional, and No Caching. After running each of these techniques through a simulation, it was

discovered that randomly selecting an episodes for propagation to the next node was the most optimal. They revisit episode propagation in [28]. Here they implement a Wireless Ad-hoc Podcasting peer-to-peer client, and measure the performance of their system with mobile devices.

[2] describes yet another method for disseminating podcasts through a wireless ad hoc network. Unlike [24] and [28], this protocol also distributes the lists of podcasts that are available to be subscribed to in the network. The authors present two methods for how nodes in the network communicate and pass podcast data: a P2P mode and a cluster mode. The cluster mode contains the concept of a cluster head, which is a node that is the strongest (in terms of attributes such as power or connectivity). The neighboring nodes only communicate with the cluster head rather than with its other neighbors that are within communication range. This organization reduces the amount of communication overhead in the network overall.

[46] investigates distributing podcast episodes to users in an urban environment. In this investigation, podcasts were either transmitted from a cellular tower to a handset, or were sent from handset to handset in an ad-hoc manner. The constraints that were experimented with were the density of cell towers, the density of users, the movement of users, and the number of users that had handsets that could actually receive from a tower. Distributing multimedia content to mobile phones is also investigated in [1].

[20] looks at the distribution of podcasts through a wireless ad hoc network of mobile devices with a focus on discovering the best method for determine how popular a podcast is within the network. It compares local knowledge (how many times a series was requested from other nodes) with second hand knowledge (how popular other nodes claim a podcast is). They created a simulator that took into consideration that mobile nodes usually live in isolated communities, and isolated communities may

have different interests and hence have different podcast popularities.

6.3 Podcast Metrics

Microsoft Research [18] examined 8000 podcast feeds from the Zune ecosystem. They examine three aspects of podcasting: the information found through the inspection of RSS feeds, the usage patterns of podcast consumers, and the dissemination pattern of podcast data through a network. For inspecting RSS feeds the authors created a crawler similar to this thesis. For inspecting usage patterns of consumers the authors inspected the proprietary Zune usage data that is reported by the Zune software to the Zune cloud service. Finally the authors consider whether or not P2P sharing of podcasts through an ad hoc network is efficient. [25] performed a survey of RSS similar to [18] but without the focus on podcasting. They provides data around RSS size and update frequency.

[4] describes a model for generating synthetic podcast traffic through a network. In order to understand the parameters that define the model, the authors polled podcast feeds on twenty minute intervals over a thirty day period. The focus of their findings was around file size and release frequency; however their sample set was small (only 875 feeds collected from now defunct websites).

6.4 Podcast Search

The authors of [34] describe a Japanese language podcast search service they created that has two innovations. First, it performs full speech recognition on each episode, and indexes the words for text based searching. Second, it provides an easy to use interface for users to fix recognition errors. These fixes are then used to improve the speech recognition creating a positive feedback loop.

[9] seeks to gain an understand of how users currently search for podcasts, how they perceive podcast search, and how they would like to search for podcasts. The authors performed two rounds of user research. First they distributed an anonymous survey, and second they performed in person interviews. Unsurprisingly, iTunes was the most popular podcast search service used by the subjects. Emphasis was made around the fact that most users thought that speech recognition of podcasts was not possible, and therefore content within an episode was not searchable.

[32] describes a text to speech system that takes a document as input and creates a series of podcast episodes as output. A heuristic is applied to the document in order to identify independent sections. Each of these independent sections (delineated by things such as bullet point, bolded titles, and numbering) becomes a podcast.

6.5 Podcast Consumption

The Pew Research Center and Edison Research both have produced studies of consumer adoption of podcasting. The Pew Internet Podcast Memo of 2008 [37] revealed 19% of internet users have download a podcast (12% in 2006). 22% of online men, 16% of online women. People with higher incomes have downloaded podcasts more than lower income earners. People with higher education have downloaded podcasts more than lower education. In 2008 a joint Arbitron and Edison paper [14] reported the percentage of the total U.S. population that is familiar with podcasting was the same at 37%. Found that 18% of the total population had listend to a podcast. Most downloadable media was consumed on a desktop computer. 71% of podcasts consumed on a PC, and 29% consumed with a portable media device. Podcast consumers are very active online purchasers, 82% versus 59% of the population at large. A follow-up report from 2009 [15] found the number of people who have

heard of podcasting increased to 43%. The stats were again updated with figures for 2010 [16].

6.6 Podcast Applications

[40] describes the planning and construction of essentially a public art fixture intended to provide easy access to audio content such as streaming radio and podcasts. The fixture takes hand gestures as input that allows the non-tech savvy user to select the podcasts they may be interested in listening to. The paper discusses the construction of the system, but does not provide any data around usage of the system by real users. [11] demonstrates the merging RSS with mapping systems. It describes a system that parses RSS for location clues and then superimposed the feed entry onto a map to give the user the ability to browse news in a geographical context. [44] describe an audio tour system that includes an animated map that helps guide users through a physical area. The application they created uses location based information to playback podcasts at pre- defined geographical locations.

CHAPTER VII

CONCLUSION AND FUTURE WORK

Users encounter frequent information quality problems in podcasting systems. The defects that are causing these problems exist despite the availability of a feed validation service provided by the World Wide Web Consortium. Therefore, we have concluded that the existing feed validator service alone is insufficient at detecting information quality defects that are specific to podcasting. To better understand the extent of this data quality problem, and to provide a mechanism for improving the quality of podcasting systems we have applied the Total Data Quality Management methodology to podcasting data.

To build an understanding of the information quality problems encountered by users of podcasting systems we performed a measurement and analysis of popular commercial podcasting platforms. To perform the measurement we constructed PodBot, an automated web agent. PodBot performed a crawl of 72,786 unique podcast feeds found in the sixteen categories of the iTunes catalog. The size and values of various feed elements were captured and summarized in order to build a model of podcast metadata. Metrics were examined over multiple dimensions, including

iTunes category and popularity. We believe this was the most comprehensive effort at modeling podcasting metadata to date.

Existing feed validation services were explored so as to understand how data quality is currently validated by podcast producers. Really Simple Syndication 2.0, the markup language specification used for syndicating podcasting content, was described. The open source Feed Validation project which is the software used by the World Wide Web Consortium for its RSS 2.0 validation services was encapsulated and included in PodBot. PodBot attempted to validate each feed from the iTunes catalog for RSS 2.0 compliance. It was found that 42% of all podcasts feeds failed this validation.

Following the Total Data Quality Management methodology we performed an analysis of information quality problems using feeds that had successfully passed validation. Information quality problems were reproduced with the Zune podcasting software from Microsoft. The quality problems demonstrated with Zune using valid feeds were classified into families of podcast specific failures. From this classification we defined a series of five data quality requirements for podcasting. We have labeled these requirements: Correctness, Uniqueness, Platform Adherence, Chronology, and Performance. From this analysis we concluded that the existing validation service is insufficient at guaranteeing a feed is free from defects that cause failures in podcasting systems.

Finally to provide a mechanism for detecting podcast specific defects in valid RSS feeds we constructed an improved validation tool - PodCop. This improved validator contains rules for enforcing the Correctness, Uniqueness, and Chorology data quality requirements. 2,594 RSS 2.0 complaint feeds were evaluated with the PodCop. 66% of these feeds were identified as being in violation of a quality requirement rule. Therefore we have demonstrated that PodCop is capable of detecting defects that

cause information quality problems is podcasting systems.

7.1 Future Work

Opportunities exist to continue building a more comprehensive model of podcasting. This thesis as well as the previous measurement attempts [4, 18] have captured metrics through the inspection of podcasting feeds. Therefore, the characteristics of the multimedia files that comprise the episodes of a podcast series are still unknown. We can envision a future version of the PodBot that downloads and inspects both the feed and all of the files linked to from within the feed. Furthermore, given that a survey of podcast episodes would need to download and inspect files that are many orders of magnitude larger than an RSS feed, we recommend the creation of a distributed client. This next generation podcast research client could run in parallel with iTunes, and inspect episodes as users naturally download them for personal consumption. This research client could be hosted by students over the course of a semester, thus preventing the overloading of any single research node.

From a data quality perspective, more can be done to understand feeds that fail RSS 2.0 validation. The version of the PodBot used in this thesis did not attempt to parse invalid feeds. The metrics captured from the various elements of a podcast feed come only from those feeds that were found to be RSS 2.0 compliant. Therefore, the characteristics of an invalid feed are largely unknown. Of particular interest would be the values of the generator element. Capturing this value would provide insight into which RSS authoring tools are producing the invalid markup.

An assumption at the time of PodBot development was that the .NET SyndicationFeed component would not necessarily be able to successfully parse invalid markup. Experiments are needed to understand the performance of this .NET component with a known set of invalid feeds. This experimentation would reveal whether

or not the SyndicationFeed is an appropriate tool for capturing podcasting metrics regardless of RSS compliance.

Finally, more effort is needed to understand how podcast feeds are unique from traditional newsfeeds. From this understanding a more refined and targeted set of podcast quality requirements can be developed. Podcasting systems are changing constantly. This thesis focused on rules appropriate to existing desktop client systems. However, podcasting systems are now made available on a variety of mobile computing platforms. Therefore work is needed to ensure the quality rules proposed here are applicable to this new context. Improving and validating the quality requirements against new platforms will enable the development of future podcast validators.

BIBLIOGRAPHY

- [1] D. Aldrich, B. Bell, and T. Batzel. Automated podcasting solution expands the boundaries of the classroom. In *Proceedings of the 34th annual ACM SIGUCCS fall conference*, SIGUCCS '06, pages 1–4, New York, NY, USA, 2006. ACM.
- [2] A. Andronache, M. R. Brust, and S. Rothkugel. Hycast- podcast discovery in mobile networks. In *Proceedings of the 3rd ACM workshop on Wireless multimedia networking and performance modeling*, WMuNeP '07, pages 27–34, New York, NY, USA, 2007. ACM.
- [3] Apple. Making a podcast. <http://www.apple.com/itunes/podcasts/specs.html>, August 2012.
- [4] A. Banerjee, M. Faloutsos, and L. Bhuyan. Profiling podcast-based content distribution. In *INFOCOM Workshops 2008, IEEE*, pages 1 –6, april 2008.
- [5] C. Batini, C. Cappiello, C. Francalanci, and A. Maurino. Methodologies for data quality assessment and improvement. *ACM Comput. Surv.*, 41(3):16:1–16:52, July 2009.
- [6] E. Berger. Podcasting in engineering education: A preliminary study of content, student attitudes, and impact. *Innovate*, 4(1), 2008.
- [7] Berkman Center for Internet & Society. Rss history. <http://cyber.law.harvard.edu/rss/rssVersionHistory.html>, June 2011.
- [8] Berkman Center for Internet & Society. Rss 2.0 at harvard law. <http://cyber.law.harvard.edu/rss/rss.html>, 2012.

- [9] J. Besser, K. Hofmann, and M. Larson. An exploratory study of user goals and strategies in podcast search. In *LWA*, pages 27–34, 2008.
- [10] H. Bui, D. Wright, C. Helm, R. Witty, P. Flynn, and D. Thain. Towards long term data quality in a large scale biometrics experiment. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, HPDC '10*, pages 565–572, New York, NY, USA, 2010. ACM.
- [11] Y.-F. R. Chen, G. Di Fabbri, D. Gibbon, S. Jora, B. Renger, and B. Wei. Geotracker: geospatial and temporal rss navigation. In *Proceedings of the 16th international conference on World Wide Web, WWW '07*, pages 41–50, New York, NY, USA, 2007. ACM.
- [12] M. Collins, C. Reynolds, C. Le, C. Varol, and C. Bayrak. Automated data verification in a format-free environment. *SIGSOFT Softw. Eng. Notes*, 31(2):1–4, Mar. 2006.
- [13] J. Duffy. How to create a podcast for e-learning: Coverage from the devlearn 2010 conference. *eLearn*, 2010(11), Nov. 2010.
- [14] Edison Research. The podcast consumer revealed. http://www.edisonresearch.com/2008_Edison_Arbitron_Podcast_Report.pdf, 2008.
- [15] Edison Research. The podcast consumer revealed 2009. http://www.edisonresearch.com/2009_Edison_Podcast_Consumer_Revealed.pdf, 2009.
- [16] Edison Research. The current state of podcasting. http://www.edisonresearch.com/2010_Edison_Podcast_Study_Data_Graphs_Only.pdf, 2010.

- [17] J. D. González Arvelo and J. P. Aspas. Design and implementation of a multimedia content delivery system for portable devices. In *Proceedings of the 3rd international conference on Mobile multimedia communications*, MobiMedia '07, pages 13:1–13:6, ICST, Brussels, Belgium, Belgium, 2007. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).
- [18] D. Gunawardena, T. Karagiannis, A. Proutiere, and M. Vojnovic. Characterizing podcast services: publishing, usage, and dissemination. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, IMC '09, pages 209–222, New York, NY, USA, 2009. ACM.
- [19] C. Hantak. Gamercast network. <http://feeds.feedburner.com/gamercastnetwork>, June 2012.
- [20] L. Hu and L. Dittmann. Reputation system for user-generated podcasting under community based mobility model. In *Proceedings of the 4th Annual International Conference on Wireless Internet*, WICON '08, pages 75:1–75:8, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).
- [21] W. Hürst, M. Welte, and S. Jung. An evaluation of the mobile usage of e-lecture podcasts. In *Proceedings of the 4th international conference on mobile technology, applications, and systems and the 1st international symposium on Computer human interaction in mobile technology*, Mobility '07, pages 16–23, New York, NY, USA, 2007. ACM.
- [22] B. L. Kurtz, J. B. Fenwick, Jr., and C. C. Ellsworth. Using podcasts and tablets in computer science. In *Proceedings of the 45th annual southeast regional conference*, ACM-SE 45, pages 484–489, New York, NY, USA, 2007. ACM.

- [23] L. Larraga and D. Coleman. Video podcasting is not as hard or as expensive as you think. In *Proceedings of the 35th annual ACM SIGUCCS fall conference, SIGUCCS '07*, pages 202–206, New York, NY, USA, 2007. ACM.
- [24] V. Lenders, M. May, G. Karlsson, and C. Wacha. Wireless ad hoc podcasting. *SIGMOBILE Mob. Comput. Commun. Rev.*, 12(1):65–67, Jan. 2008.
- [25] H. Liu, V. Ramasubramanian, and E. G. Sirer. Client behavior and feed characteristics of rss, a publish-subscribe system for web micronews. In *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement, IMC '05*, pages 3–3, Berkeley, CA, USA, 2005. USENIX Association.
- [26] S. E. Madnick, R. Y. Wang, Y. W. Lee, and H. Zhu. Overview and framework for data and information quality research. *J. Data and Information Quality*, 1(1):2:1–2:22, June 2009.
- [27] D. J. Malan. Podcasting computer science e-1. *SIGCSE Bull.*, 39(1):389–393, Mar. 2007.
- [28] M. May, V. Lenders, G. Karlsson, and C. Wacha. Wireless opportunistic podcasting: implementation and design tradeoffs. In *Proceedings of the second ACM workshop on Challenged networks, CHANTS '07*, pages 75–82, New York, NY, USA, 2007. ACM.
- [29] Microsoft. Providing content for zune. <http://zune.net/en-US/support/usersguide/podcasts/create.htm>, 2012.
- [30] Microsoft. Rss: Publishing news via xml. <http://msdn.microsoft.com/en-us/library/ms947599.aspx>, August 2012.
- [31] Microsoft. Syndicationfeed class. <http://msdn.microsoft.com/en-us/library/system.servicemodel.syndication.syndicationfeed.aspx>, August 2012.

- [32] G. Mori, M. C. Buzzi, M. Buzzi, and B. Leporini. Structured audio podcasts via web text-to-speech system. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 1281–1284, New York, NY, USA, 2010. ACM.
- [33] National Public Radio. Npr: Hourly news summary podcast. <http://www.npr.org/rss/podcast.php?id=500005>, 2012.
- [34] J. Ogata and M. Goto. Podcastle: a spoken document retrieval system for podcasts and its performance improvement by anonymous user contributions. In *Proceedings of the third workshop on Searching spontaneous conversational speech*, SSCS '09, pages 37–38, New York, NY, USA, 2009. ACM.
- [35] P. R. Ormond. Podcasting enhances learning. *J. Comput. Sci. Coll.*, 24(1):232–238, Oct. 2008.
- [36] A. Page, K. Johnston, B. Rollison, and L. Finnel. *How We Test Software at Microsoft*. Microsoft Press, 2009.
- [37] Pew Research Center. Pew internet project podcast 2008 memo. http://www.pewinternet.org/media/Files/Reports/2008/PIP_Podcast_2008_Memo.pdf, 2008.
- [38] M. Pilgrim and S. Ruby. Feed validator. <http://feedvalidator.sourceforge.net/>, August 2012.
- [39] RSS Advisory Board. Rss advisory board charter. <http://www.rssboard.org/charter>, August 2012.
- [40] E. Rubegni, J. Brunk, M. Caporali, and A. Rizzo. Wi-roni: a gesture tangible interface for experiencing internet content in public spaces. In *Proceedings of*

the 2007 workshop on Tagging, mining and retrieval of human related activity information, TMR '07, pages 15–22, New York, NY, USA, 2007. ACM.

- [41] O. A. Schulte, T. Wunden, and A. Brunner. Replay: an integrated and open solution to produce, handle, and distribute audio-visual (lecture) recordings. In *Proceedings of the 36th annual ACM SIGUCCS fall conference: moving mountains, blazing trails*, SIGUCCS '08, pages 195–198, New York, NY, USA, 2008. ACM.
- [42] M. Tsagkias, M. Larson, and M. de Rijke. Predicting podcast preference: An analysis framework and its application. *J. Am. Soc. Inf. Sci. Technol.*, 61(2):374–391, Feb. 2010.
- [43] M. Tsagkias, M. Larson, W. Weerkamp, and M. de Rijke. Podcred: a framework for analyzing podcast preference. In *Proceedings of the 2nd ACM workshop on Information credibility on the web*, WICOW '08, pages 67–74, New York, NY, USA, 2008. ACM.
- [44] K. Tsuruoka and M. Arikawa. An authoring tool for urban audio tours with animated maps. In *Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology*, ACE '08, pages 330–333, New York, NY, USA, 2008. ACM.
- [45] B. J. Voyer and C. W. Crane. Using new media to improve self-help for clients and staff. In *Proceedings of the 38th annual fall conference on SIGUCCS*, SIGUCCS '10, pages 235–240, New York, NY, USA, 2010. ACM.
- [46] V. Vukadinović and G. Karlsson. Spectral efficiency of mobility-assisted podcasting in cellular networks. In *Proceedings of the Second International Workshop*

on Mobile Opportunistic Networking, MobiOpp '10, pages 51–57, New York, NY, USA, 2010. ACM.

- [47] R. Y. Wang. A product perspective on total data quality management. *Commun. ACM*, 41(2):58–65, Feb. 1998.
- [48] T. B. Wolff. Podcasting made simple. In *Proceedings of the 34th annual ACM SIGUCCS fall conference*, SIGUCCS '06, pages 413–418, New York, NY, USA, 2006. ACM.
- [49] World Wide Web Consortium. About w3c. <http://www.w3.org/Consortium/>, 2012.
- [50] World Wide Web Consortium. Feed validation service. <http://validator.w3.org/feed/>, 2012.

APPENDIX

Listing 1: Example .NET Class SyndicationFeed Usage

```

1 using System;
2 using System.Collections.Generic;
3 using System.Collections.ObjectModel;
4 using System.ServiceModel.Syndication;
5 using System.Xml;
6 using System.Xml.Linq;
7
8 namespace SyndicationExample {
9     class Program
10    {
11        static void Main(string[] args) {
12            string feed = args[0];
13            XmlReader xmlReader = XmlReader.Create(feed);
14            SyndicationFeed syndicationFeed = SyndicationFeed.Load(
15                xmlReader);
16            PrintAttributeExtensions(syndicationFeed.AttributeExtensions
17                );
18            PrintAuthors(syndicationFeed.Authors);
19            PrintBaseUri(syndicationFeed.BaseUri);
20            PrintCategories(syndicationFeed.Categories);
21            PrintContributors(syndicationFeed.Contributors);
22            PrintCopyright(syndicationFeed.Copyright);
23            PrintDescription(syndicationFeed.Description);
24            PrintElementExtensions(syndicationFeed.ElementExtensions);
25            PrintGenerator(syndicationFeed.Generator);
26            PrintId(syndicationFeed.Id);
27            PrintImageUrl(syndicationFeed.ImageUrl);
28            PrintLanguage(syndicationFeed.Language);
29            PrintLastUpdatedTime(syndicationFeed.LastUpdatedTime);
30            PrintLinks(syndicationFeed.Links);
31            PrintTitle(syndicationFeed.Title);
32            PrintItems(syndicationFeed.Items);
33        }
34
35        static void PrintAttributeExtensions(Dictionary<XmlQualifiedName
36            , String> attributeExtensions)
37        {
38            if (attributeExtensions.Count == 0)
39            {
40                Console.WriteLine("AttributeExtensions_:"); return;
41            }
42            foreach (KeyValuePair<XmlQualifiedName, String> keyValuePair
43                in attributeExtensions) {
44                Console.WriteLine("AttributeExtensions_{0}_{1}",
45                    keyValuePair.Key, keyValuePair.Value);
46            }
47        }
48
49        static void PrintAuthors(Collection<SyndicationPerson> authors)
50        {
51            if (authors.Count == 0) {
52                Console.WriteLine("Authors_:");
53                return;
54            }
55            foreach (SyndicationPerson syndicationPerson in authors) {
56                Console.WriteLine("Authors_{0}_{1}_{2}",
57                    syndicationPerson.Name, syndicationPerson.Email,

```

```

        syndicationPerson.Uri);
51     }
52 }
53
54 static void PrintBaseUri(Uri baseUri) {
55     if (baseUri == null)
56     {
57         Console.WriteLine("BaseUri_:");
58         return;
59     }
60     Console.WriteLine("BaseUri_:_{0}", baseUri.ToString());
61 }
62
63 static void PrintCategories(Collection<SyndicationCategory>
64 categories) {
65     if (categories.Count == 0) {
66         Console.WriteLine("Categories_:");
67         return;
68     }
69     foreach (SyndicationCategory syndicationCategory in
70 categories) {
71         Console.WriteLine("Categories_:_{0},_{1},_{2}",
72 syndicationCategory.Label, syndicationCategory.Name,
73 syndicationCategory.Scheme);
74     }
75 }
76
77 static void PrintContributors(Collection<SyndicationPerson>
78 contributors) {
79     if (contributors.Count == 0)
80     {
81         Console.WriteLine("Contributors_:");
82         return;
83     }
84     foreach (SyndicationPerson syndicationPerson in contributors
85 ) {
86         Console.WriteLine("Contributors_:_{0},_{1},_{2}",
87 syndicationPerson.Name, syndicationPerson.Email,
88 syndicationPerson.Uri.ToString());
89     }
90 }
91
92 static void PrintCopyright(TextSyndicationContent copyright) {
93     if (copyright == null) {
94         Console.WriteLine("Copyright_:");
95         return;
96     }
97     Console.WriteLine("Copyright_:_{0}", copyright.Text);
98 }
99
100 static void PrintDescription(TextSyndicationContent description)
101 {
102     if (description == null)
103     {
104         Console.WriteLine("Description_:");
105         return;
106     }
107     Console.WriteLine("Description_:_{0}", description.Text);

```

```

99     }
100
101     static void PrintElementExtensions(
102         SyndicationElementExtensionCollection elementExtensions)
103     {
104         if (elementExtensions.Count == 0) {
105             Console.WriteLine("ElementExtensions_:");
106             return;
107         }
108         foreach (SyndicationElementExtension
109             syndicationElementExtension in elementExtensions) {
110             XElement xElement = syndicationElementExtension.
111                 GetObject<XElement>();
112             Console.Write("ElementExtensions_:_{0},_{1},_{2}",
113                 syndicationElementExtension.OuterName,
114                 syndicationElementExtension.OuterNamespace, xElement
115                 .Value);
116             if (xElement.HasAttributes)
117             {
118                 foreach (XAttribute xAttribute in xElement.
119                     Attributes()) {
120                     Console.Write("_{0},_{1}", xAttribute.Value,
121                         xAttribute.Name);}
122             }
123             Console.WriteLine();
124         }
125     }
126
127     static void PrintGenerator(string generator) {
128         Console.WriteLine("Generator_:_{0}", generator);
129     }
130
131     static void PrintId(string id) {
132         Console.WriteLine("Id_:_{0}", id);
133     }
134
135     static void PrintImageUrl(Uri imageUrl) {
136         if (imageUrl == null)
137         {
138             Console.WriteLine("ImageUrl_:");
139             return;
140         }
141         Console.WriteLine("ImageUrl_:_{0}", imageUrl.ToString());
142     }
143
144     static void PrintLanguage(string language) {
145         Console.WriteLine("Language_:_{0}", language);
146     }
147
148     static void PrintLastUpdatedTime(DateTimeOffset lastUpdatedTime)
149     {
150         Console.WriteLine("LastUpdateTime_:_{0}", lastUpdatedTime.
151             ToString());
152     }
153
154     static void PrintLinks(Collection<SyndicationLink> links) {
155         if (links.Count == 0)
156         {

```

```

147         Console.WriteLine("Links_:");
148         return;
149     }
150     foreach (SyndicationLink syndicationLink in links) {
151         Console.WriteLine("Links_:_{0},_{1},_{2},_{3},_{4}",
            syndicationLink.Title, syndicationLink.Length,
            syndicationLink.MediaType, syndicationLink.
            RelationshipType, syndicationLink.Uri.ToString());
152     }
153 }
154
155 static void PrintTitle(TextSyndicationContent title) {
156     if (title == null)
157     {
158         Console.WriteLine("Title_:");
159         return;
160     }
161     Console.WriteLine("Title_:_{0}", title.Text);
162 }
163
164 static void PrintItems(IEnumerable<SyndicationItem> items) {
165     foreach (SyndicationItem syndicationItem in items) {
166         Console.WriteLine();
167         Console.WriteLine("
            =====");
168         Console.WriteLine("Item_:");
169         PrintAttributeExtensions(syndicationItem.
            AttributeExtensions);
170         PrintAuthors(syndicationItem.Authors);
171         PrintBaseUri(syndicationItem.BaseUri);
172         PrintCategories(syndicationItem.Categories);
173         PrintContent(syndicationItem.Content);
174         PrintContributors(syndicationItem.Contributors);
175         PrintCopyright(syndicationItem.Copyright);
176         PrintElementExtensions(syndicationItem.ElementExtensions
            );
177         PrintId(syndicationItem.Id);
178         PrintLastUpdatedTime(syndicationItem.LastUpdatedTime);
179         PrintLinks(syndicationItem.Links);
180         PrintPublishDate(syndicationItem.PublishDate);
181         PrintSummary(syndicationItem.Summary);
182         PrintTitle(syndicationItem.Title);
183     }
184 }
185
186 static void PrintContent(SyndicationContent content) {
187     if (content == null)
188     {
189         Console.WriteLine("Content_:");
190         return;
191     }
192     Console.WriteLine("Content_:_{0}", content.Type);
193 }
194
195 static void PrintPublishDate(DateTimeOffset publishDate) {
196     Console.WriteLine("PublishDate_:_{0}", publishDate.ToString
        ());
197 }

```

```
198
199     static void PrintSummary(TextSyndicationContent summary) {
200         if (summary == null)
201         {
202             Console.WriteLine("Summary_:");
203             return;
204         }
205         Console.WriteLine("Summary_:_{0}", summary.Text);
206     }
207 }
208 }
```

Element	Description	Example
title	The name of the channel. It's how people refer to your service. If you have an HTML website that contains the same information as your RSS file, the title of your channel should be the same as the title of your website.	GoUpstate.com News Headlines
link	The URL to the HTML website corresponding to the channel.	http://www.goupstate.com/
description	Phrase or sentence describing the channel.	The latest news from GoUpstate.com, a Spartanburg Herald-Journal Web site.
language	The language the channel is written in. This allows aggregators to group all Italian language sites, for example, on a single page. A list of allowable values for this element, as provided by Netscape, is here. You may also use values defined by the W3C.	en-us

copyright	Copyright notice for content in the channel.	Copyright 2002, Spartanburg Herald-Journal
managingEditor	Email address for person responsible for editorial content.	geo@herald.com (George Matesky)
webMaster	Email address for person responsible for technical issues relating to channel.	betty@herald.com (Betty Guernsey)
pubDate	The publication date for the content in the channel. For example, the New York Times publishes on a daily basis, the publication date flips once every 24 hours. That's when the pubDate of the channel changes. All date-times in RSS conform to the Date and Time Specification of RFC 822, with the exception that the year may be expressed with two characters or four characters (four preferred).	Sat, 07 Sep 2002 00:00:01 GMT
lastBuildDate	The last time the content of the channel changed.	Sat, 07 Sep 2002 09:42:31 GMT

category	Specify one or more categories that the channel belongs to. Follows the same rules as the <code><item></code> -level category element.	<code><category>Newspapers </category></code>
generator	A string indicating the program used to generate the channel.	MightyInHouse Content System v2.3
docs	A URL that points to the documentation for the format used in the RSS file. It's probably a pointer to this page. It's for people who might stumble across an RSS file on a Web server 25 years from now and wonder what it is.	<code>http://blogs.law.harvard.edu/tech/rss</code>
cloud	Allows processes to register with a cloud to be notified of updates to the channel, implementing a lightweight publish-subscribe protocol for RSS feeds.	<code><cloud domain='rpc.sys.com' port='80' path='/RPC2' registerProcedure='pingMe' protocol="soap"/></code>

ttl	ttl stands for time to live. It's a number of minutes that indicates how long a channel can be cached before refreshing from the source.	<code><ttl>60</ttl></code>
image	Specifies a GIF, JPEG or PNG image that can be displayed with the channel.	
rating	The PICS rating for the channel.	
textInput	Specifies a text input box that can be displayed with the channel.	
skipHours	A hint for aggregators telling them which hours they can skip.	
skipDays	A hint for aggregators telling them which days they can skip.	

Table 33: RSS 2.0 Channel Elements

# Feeds	Error
2405	Undefined root element: xhtml:html
2084	Undefined root element: script
2024	XML parsing error: <unknown>:6:41: not well-formed (invalid token)
2012	link must be a full and valid URL
1897	Invalid email address
1480	Incorrect day of week
1394	pubDate must be an RFC-822 date-time
1254	Invalid character in a URI
1184	Undefined channel element: itunes:link
1097	Undefined root element: html
1093	guid values must not be duplicated within a feed
1023	Unexpected Text
1010	width must be between 1 and 144
867	Missing channel element: description
657	Invalid duration
652	url must be a full URL
637	lastBuildDate must be an RFC-822 date-time
628	language must be an ISO-639 language code
601	XML parsing error: <unknown>:23:35: not well-formed (invalid token)
567	href must be a full URL
546	Incorrect day of week (2 occurrences)
502	Undefined channel element: itunes:complete
499	Missing image element: link
490	Undefined channel element: itunes:provider

409	Missing image element: title
386	Missing atom:link with rel="self"
360	Missing itunes:image attribute: href (100 occurrences)
358	Missing itunes:owner element: itunes:email
357	Unexpected Text (100 occurrences)
342	Educational Technology is not one of the predefined iTunes categories or sub-categories
342	Missing itunes:image attribute: href
334	pubDate must be an RFC-822 date-time (2 occurrences)
320	length attribute of enclosure must be a positive integer
320	Undefined item element: itunes:category
310	XML parsing error: <unknown>:1:0: syntax error
306	Undefined channel element: background1
306	Undefined channel element: background2
306	Undefined channel element: displayrows
306	Undefined channel element: foreground
306	Undefined channel element: lg:headerimage
306	Undefined channel element: link:color
306	Undefined channel element: md:headerimage
306	Undefined channel element: show:xmltag
306	Undefined channel element: sm:headerimage
306	Undefined channel element: textsize
306	Undefined channel element: title:bg
306	Undefined channel element: title:fg

292	Podcasting is not one of the predefined iTunes categories or subcategories
-----	--

Table 34: The Top 50 RSS 2.0 Violations in Podcast Feeds

Element	Description	Example
title	The title of the item.	Venice Film Festival Tries to Quit Sinking.
link	The URL of the item.	http://nytimes.com/2004/12/07FEST.html
description	The item synopsis.	Some of the most heated chatter at the Venice Film Festival this week was about the way that the arrival of the stars at the Palazzo del Cinema was being staged.
author	Email address of the author of the item.	oprah@oxygen.net
category	Includes the item in one or more categories.	
comments	URL of a page for comments relating to the item.	http://www.myblog.org/cgi-local/mt/mt-comments.cgi?entry_id=290
enclosure	Describes a media object that is attached to the item.	
guid	A string that uniquely identifies the item.	http://inessential.com/2002/09/01.php#a2
pubDate	Indicates when the item was published.	Sun, 19 May 2002 15:21:36 GMT
source	The RSS channel that the item came from.	

Table 35: The RSS 2.0 Item Elements

Domain	# Feeds	Feed %
feedburner.com	17616	24.25%
libsyn.com	3753	5.17%
podbean.com	2481	3.42%
podomatic.com	1898	2.61%
blip.tv	1583	2.18%
blogtalkradio.com	1193	1.64%
mypodcast.com	852	1.17%
talkshoe.com	831	1.14%
mac.com	755	1.04%
apple.com	652	0.90%
me.com	565	0.78%
mevio.com	512	0.72%
podiobooks.com	438	0.60%
librivox.org	409	0.56%
gcast.com	310	0.44%
amazonaws.com	305	0.42%
hipcast.com	291	0.40%
bbc.co.uk	276	0.38%
libsynpro.com	271	0.37%
podcastmachine.com	237	0.33%
podspot.de	213	0.29%
audioacrobat.com	211	0.29%
dw-world.de	169	0.23%
sermonaudio.com	150	0.21%
peerviewpress.com	150	0.21%
npr.org	146	0.20%
odeo.com	138	0.19%
jellycast.com	126	0.17%
revision3.com	124	0.17%
go.com	122	0.17%

Table 36: The Top 30 Domains Hosting Podcast Feeds

Feed	Popularity
http://feeds.thisamericanlife.org/talpodcast	1.0000000000
http://feeds.feedburner.com/comedycentral/standup	0.2909667500
http://feeds.feedburner.com/themothpodcast	0.2301070800
http://americanpublicmedia.publicradio.org/podcasts/xml/prairie_home_companion/news_from_lake_wobegon.xml	0.1780055500
http://www.qdnw.com/grammar.xml	0.1394481800
http://feeds.feedburner.com/TEDTalks_video	0.1343486000
http://selectedshortspri.pri.libsynpro.com/rss	0.1080078700
http://wtfpod.libsyn.com/rss	0.1031321360
http://feeds.newyorker.com/services/rss/feeds/fiction_podcast.xml	0.0986269040
http://www.gcast.com/u/dane_cook/main.xml	0.0960611200

Table 37: The Top 10 Most Popular Arts Podcasts

Feed	Popularity
http://www.npr.org/rss/podcast.php?id=510289	1.0000000000
http://www.daveramsey.com/media/audio/podcast/podcast_itunes.xml	0.9912208000
http://podcast.cnbc.com/mmpodcast/lightninground.xml	0.5800561000
http://feeds.wsjonline.com/wsj/podcast_wall_street_journal_this_morning	0.5766480000
http://feeds.americanpublicmedia.org/MarketplacePodcast	0.4836761000
http://feeds.harvardbusiness.org/harvardbusiness/ideacast	0.4281649600
http://www.apple.com/podcasts/quicktips/apple-quick-tip-of-the-week.xml	0.3907846500
http://feeds.wnyc.org/onthemedial	0.3906947700
http://www.qdnw.com/money.xml	0.3217286000
http://www.businessweek.com/search/podcasts/cover_stories.rss	0.3042813000

Table 38: The Top 10 Most Popular Business Podcasts

Feed	Popularity
http://feeds.feedburner.com/comedycentral/standup	1.0000000000
http://www.theonion.com/feeds/radionews/	0.9951623000
http://feeds.feedburner.com/boyt	0.9634709400
http://podcast.happytreefriends.com/htfrss.xml	0.8517678400
http://feeds.feedburner.com/themothpodcast	0.7939343000
http://feeds.feedburner.com/TheAdamCarollaPodcast	0.7093609600
http://americanpublicmedia.publicradio.org/podcasts/xml/prairie_home_companion/news_from_lake_wobegon.xml	0.6135832700
http://feeds.theonion.com/OnionNewsNetwork	0.6104997400
http://feeds.feedburner.com/vh1_bestweekever	0.5966233000
http://podcast.rickygervais.com/podcast_new.xml	0.5680127000

Table 39: The Top 10 Most Popular Comedy Podcasts

Feed	Popularity
http://feeds.wnyc.org/radiolab	1.0000000000
http://feeds.feedburner.com/coffeebreakspanish	0.7231777000
http://www.qdnw.com/grammar.xml	0.5882143400
http://feeds.feedburner.com/TEDTalks_video	0.5696028000
http://survivalspanish.libsyn.com/rss	0.2955899500
http://writersalmanac.publicradio.org/podcast/feed.php	0.2818662200
http://www.frenchpodclass.com/rss	0.2536054800
http://feeds.feedburner.com/TEDTalks_audio	0.2484416400
http://feeds.feedburner.com/coffeebreakfrench	0.2403352300
http://podcast.discoverspanish.com/audiolessons/	0.2268316600

Table 40: The Top 10 Most Popular Education Podcasts

Feed	Popularity
http://www.npr.org/rss/podcast.php?id=35	1.0000000000
http://www.npr.org/rss/podcast.php?id=510208	0.5664123000
http://www.g4tv.com/xplay/podcasts/6/G4_TV_XPlay_Video_Podcast.xml	0.2471247800
http://www.myextralife.com/ftp/radio/instance_rss.xml	0.0798286050
http://www.npr.org/rss/podcast.php?id=4473090&uid=n1qe4e85742c986fdb81d2d38ffa0d5d53	0.0723763400
http://feeds.tipsfromthetopfloor.com/tftf	0.0723261300
http://feeds.feedburner.com/learningguitarnow	0.0639601950
http://feeds.feedburner.com/GSGPodcasts	0.0632954500
http://feeds.feedburner.com/1UP/1upShow	0.0590947940
http://feeds.ign.com/ignfeeds/podcasts/games/	0.0562915300

Table 41: The Top 10 Most Popular Games & Hobbies Podcasts

Feed	Popularity
http://www.democracynow.org/podcast.xml	1.0000000000
http://feeds.feedburner.com/walkintheword/wxZf	0.4419155700
http://www.democracynow.org/podcast-video.xml	0.4115143400
http://feeds.feedburner.com/BestOfTheLeftPodcast	0.3253019000
http://feeds.pbs.org/pbs/frontlineworld	0.2984872000
http://www.qdnow.com/legal.xml	0.2432659000
http://georgewbush-whitehouse.archives.gov/rss/radioaddress.xml	0.2253754000
http://www.mevio.com/feeds/noagenda.xml	0.1611784100
http://feeds.feedburner.com/blasttheright	0.1356372700
http://www.makochemedia.com/files/tjh.xml	0.1316714900

Table 42: The Top 10 Most Popular Government & Organizations Podcasts

Feed	Popularity
http://www.daveramsey.com/media/audio/podcast/podcast_itunes.xml	1.0000000000
http://podrunner.wm.wizzard.tv/rss	0.7293028000
http://www.oprah.com/podcasts/anewearth.xml	0.7089863000
http://feeds.thestranger.com/stranger/savage	0.6077437000
http://feeds.feedburner.com/yogamazing	0.4416526300
http://fitpod.libsyn.com/rss	0.3974276800
http://americanpublicmedia.publicradio.org/podcasts/xml/splendid_table/kitchen_questions.xml	0.3646704600
http://feeds.feedburner.com/zencast	0.2574263200
http://feeds.feedburner.com/yogadownload	0.2571052000
http://www.ullreys.com/robert/Podcasts/page4/files/rss.xml	0.2528306500

Table 43: The Top 10 Most Popular Health Podcasts

Feed	Popularity
http://americanpublicmedia.publicradio.org/podcasts/xml/prairie_home_companion/news_from_lake_wobegon.xml	1.0000000000
http://www.daveramsey.com/media/audio/podcast/podcast_itunes.xml	0.9131060000
http://www.mugglenet.com/mugglecast/mugglecast.rss	0.4700909000
http://podcasts.sesamestreet.org/SesameStreetPodcast	0.4654681400
http://americanpublicmedia.publicradio.org/podcasts/xml/splendid_table/kitchen_questions.xml	0.3313115800
http://feeds.feedburner.com/ellenshow.rss	0.2686919600
http://www.the-leaky-cauldron.org/podcasts/pottercast.rss	0.2661548300
http://radio.disney.go.com/podcasts/itunes/radio_disney_now.xml	0.2446642400
http://www.qdnw.com/manners.xml	0.2328243700
http://feeds.feedburner.com/Storynory	0.2114372800

Table 44: The Top 10 Most Popular Kids & Family Podcasts

Feed	Popularity
http://www.npr.org/rss/podcast.php?id=510019&uid=n1qe4e85742c986fdb81d2d38ffa0d5d53	1.0000000000
http://feeds.feedburner.com/tiestos_club_life	0.5682367000
http://podrunner.wm.wizzard.tv/rss	0.5563007600
http://www.ringtonefeeder.com/promo/freedemo.xml	0.4569853000
http://www.npr.org/rss/podcast.php?id=510253	0.4239159500
http://podcast.armadamusic.com/asot/podcast.xml	0.3650835200
http://feeds.kexp.org/kexp/songoftheday	0.3344156000
http://feeds.feedburner.com/IndiefeedAlt/modernRock	0.3234398700
http://fitpod.libsyn.com/rss	0.3040501200
http://feeds.kcrw.com/kcrw/mb	0.2935790700

Table 45: The Top 10 Most Popular Music Podcasts

Feed	Popularity
http://feeds.thisamericanlife.org/talpodcast	1.0000000000
http://www.hbo.com/podcasts/billmaher/podcast.xml	0.3066271800
http://www.sciencefriday.com/audio/scifriaudio.xml	0.2158997000
http://www.npr.org/rss/podcast.php?id=1090&uid=n1qe4e85742c986fdb81d2d38ffa0d5d53	0.1869892200
http://feeds.feedburner.com/economist/audio_all	0.1859505200
http://podcastfeeds.nbcnews.com/audio/podcast/MSNBC-NN-NETCAST-M4V.xml	0.1712706000
http://www.npr.org/rss/podcast.php?id=510289	0.1649607100
http://downloads.bbc.co.uk/podcasts/worldservice/globalnews/rss.xml	0.1579105300
http://www.npr.org/rss/podcast/TOTNPodcast.xml	0.1534835700
http://www.cbsradionewsfeed.com/rss.php?id=90&ud=512	0.1513136200

Table 46: The Top 10 Most Popular News & Politics Podcasts

Feed	Popularity
http://www.joelosteen.com/_vti_bin/JOMHelper.aspx/GetPodcastAudio	1.0000000000
http://www.oprah.com/podcasts/anewearth.xml	0.7184295000
http://www.joelosteen.com/_vti_bin/JOMHelper.aspx/GetPodcastVideo	0.5549426700
http://feeds.feedburner.com/joycemeyer/SFiE	0.4629326000
http://feeds.feedburner.com/dailyaudiobible	0.4399391000
http://feeds.marshall.com/marshall/mark-driscoll/audio	0.4215766800
http://being.publicradio.org/podcast/podcast.xml	0.3737928000
http://feeds2.feedburner.com/DGSermonAudio	0.3613710700
http://rss.streamos.com/streamos/rss/genfeed.php?feedid=17&groupname=itm	0.3290070300
http://feeds.feedburner.com/joycemeyer/IEAM	0.3111004500

Table 47: The Top 10 Most Popular Religion & Spirituality Podcasts

Feed	Popularity
http://feeds.wnyc.org/radiolab	1.0000000000
http://www.sciencefriday.com/audio/scifriaudio.xml	0.9220162600
http://www.howstuffworks.com/podcasts/brainstuff.rss	0.3151734200
http://www.scientificamerican.com/podcast/sciam_podcast_i.x ml	0.2894540400
http://www.scientificamerican.com/podcast/sciam_podcast_i_d.xml	0.2858437000
http://feeds.feedburner.com/TEDTalks_audio	0.2481027400
http://www.spitzer.caltech.edu/resource_list/6-Hidden-Universe-NASA-s-Spitzer-Space-Telescope?def=hi&format=xml	0.2443366800
http://feeds.feedburner.com/cnet/buzzoutloud	0.2346226300
http://feeds.feedburner.com/TedtalksHD	0.1986399300
http://blog.makezine.com/archive/category/make_podcast/feed	0.1882510300

Table 48: The Top 10 Most Popular Science & Medicine Podcasts

Feed	Popularity
http://feeds.thisamericanlife.org/talpodcast	1.0000000000
http://www.npr.org/rss/podcast.php?id=13	0.4450369200
http://www.howstuffworks.com/podcasts/stuff-you-should-know.rss	0.3427562400
http://feeds.wnyc.org/radiolab	0.2345489900
http://www.howstuffworks.com/podcasts/stuff-you-missed-in-history-class.rss	0.1974614100
http://feeds.feedburner.com/freakonomicsradio	0.1681365400
http://www.npr.org/rss/podcast.php?id=510289	0.1652642300
http://feeds.feedburner.com/TEDTalks_video	0.1343809700
http://feeds.americanpublicmedia.org/MarketplacePodcast	0.0796713500
http://www.howstuffworks.com/podcasts/brainstuff.rss	0.0737197500

Table 49: The Top 10 Most Popular Society & Culture Podcasts

Feed	Popularity
http://sports.espn.go.com/espnradio/podcast/feeds/itunes/podcast?id=2406595	1.0000000000
http://sports.espn.go.com/espnradio/podcast/feeds/itunes/podcast?id=2864045	0.5870259400
http://sports.espn.go.com/espnradio/podcast/feeds/itunes/podcast?id=2090484	0.4460636400
http://sports.espn.go.com/espnradio/podcast/feeds/itunes/podcast?id=2445552	0.4256554800
http://sports.espn.go.com/espnradio/podcast/feeds/itunes/podcast?id=2942325	0.3060259500
http://fitpod.libsyn.com/rss	0.2930497800
http://sports.espn.go.com/espnradio/podcast/feeds/itunes/podcast?id=2386164	0.2423153500
http://sports.espn.go.com/espnradio/podcast/feeds/itunes/podcast?id=2839445	0.2399011900
http://www.danpatrick.com/podcasts/feed/	0.2310778800
http://sports.espn.go.com/espnradio/podcast/feeds/itunes/podcast?id=2544457	0.2058922600

Table 50: The Top 10 Most Popular Sports & Recreation Podcasts

Feed	Popularity
http://leoville.tv/podcasts/twit.xml	1.0000000000
http://feeds.feedburner.com/TEDTalks_video	0.7324087000
http://itstreaming.apple.com/podcasts/apple_keynotes/apple_keynotes.xml	0.6169259000
http://www.ringtonefeeder.com/promo/freedemo.xml	0.5381654500
http://leoville.tv/podcasts/kfi.xml	0.4801688800
http://www.npr.org/rss/podcast.php?id=1019&uid=n1qe4e85742c986fdb81d2d38ffa0d5d53	0.4617037500
http://blip.tv/photoshop-user-tv/rss/itunes	0.4526480700
http://www.howstuffworks.com/podcasts/brainstuff.rss	0.4035999800
http://revision3.com/dignation/feed/quicktime-small/	0.3888102800
http://www.scientificamerican.com/podcast/sciam_podcast_i.xml	0.3706646300

Table 51: The Top 10 Most Popular Technology Podcasts

Feed	Popularity
http://feeds.feedburner.com/comedycentral/standup	1.0000000000
http://feeds.feedburner.com/vh1_bestweekever	0.5968024000
http://podcast.rickygervais.com/podcast_new.xml	0.5627677400
http://www.discovery.com/radio/xml/discovery_video.xml	0.4733996000
http://www.g4tv.com/xplay/podcasts/6/G4_TV__XPlay_Video_Podcast.xml	0.4669278000
http://dogma.vo.llnwd.net/o25/NewMoon/ipodTest/USM.ItunesClip.xml	0.4252564300
http://www.oprah.com/podcasts/anewearth.xml	0.3994750700
http://wtfpod.libsyn.com/rss	0.3479579000
http://www.gcast.com/u/dane_cook/main.xml	0.3303963500
http://www.g4tv.com/attackoftheshow/podcasts/5/Attack_of_the_Show_Daily_Video_Podcast.xml	0.3254902000

Table 52: The Top 10 Most Popular TV & Film Podcasts

MIME Type	Episode %
audio/mpeg	77.103%
video/mp4	5.536%
video/x-m4v	4.305%
NO MEDIA	4.161%
audio/x-m4a	2.657%
video/quicktime	1.428%
audio/mp3	0.915%
audio/mp4	0.758%
application/pdf	0.586%
video/mpeg	0.328%
video/x-mp4	0.282%
video/m4v	0.262%
audio/x-mp3	0.210%
application/octet-stream	0.193%
audio/x-mpeg	0.148%
video/mov	0.127%
audio/mpeg3	0.106%
x-audio/mp3	0.104%
audio/aac	0.103%
application/x-shockwave-flash	0.101%
video/x-m4a	0.071%
text/plain	0.036%
image/jpeg	0.035%
audio/x-wav	0.034%
video/x-flv	0.028%
audio/x-m4b	0.027%
text/html	0.022%
video/x-ms-wmv	0.021%
audio/m4a	0.020%

Table 53: The Top 30 Podcast Episode Formats

MIME Type	Mean	Median	Min	Max
audio/mpeg	98121994.54483	17387947	0	9.22337E+18
video/mp4	142006132.98446	36610336	0	2.415E+11
video/x-m4v	80987019.15629	30209162	0	16388000000
audio/x-m4a	33270999.27017	19772035	0	32108669329
video/quicktime	71296570.38464	26035364	0	4294967295
audio/mp3	26484489.81866	13238272	0	960000000
audio/mp4	32719813.58931	23886858	0	591085298
application/pdf	1328736.00124	175411	0	127919864
video/mpeg	60612693.64153	20948741	0	2213648319
video/x-mp4	55248559.56602	24588389	0	1670899837
video/m4v	73342508.63629	33671513	0	3451031149
audio/x-mp3	23168443.09555	9700724	0	857735168
application/octet-stream	46400826.44470	26328896	0	931969929
audio/x-mpeg	20682820.18931	14484375	0	153646134
video/mov	15175940.40630	107603	0	3763358544
audio/mpeg3	26547860.48566	21990172	3	650635927
x-audio/mp3	17421659.04184	15750000	0	484372960
audio/aac	?22010393.84437	16856858	0	151093265
application/x-shockwave-flash	78912.95986	1190	0	34174435
video/x-m4a	50313285.44569	43678734	65437	313949767
text/plain	32277626.01603	19649143	0	481809503
image/jpeg	139662.65975	25638	0	7500000
audio/x-wav	?57301055.82251	13890579	0	645225704
video/x-flv	51127452.18110	24192098	0	402078442
audio/x-m4b	40907862.66220	37530510	1006318	216459117
text/html	3507573.34564	0	0	141385409
video/x-ms-wmv	129657421.98625	35801071	0	1137945894
audio/m4a	34152054.43885	27921275	0	298871759

Table 54: Sizes of the Top 30 Podcast Formats

Namespace	Feeds %
http://www.itunes.com/dtds/podcast-1.0.dtd	100.000%
http://search.yahoo.com/mrss/	43.917%
http://purl.org/dc/elements/1.1/	37.027%
http://rssnamespace.org/feedburner/ext/1.0	32.931%
http://purl.org/rss/1.0/modules/content/	27.861%
http://wellformedweb.org/CommentAPI/	26.484%
http://purl.org/rss/1.0/modules/syndication/	19.744%
http://purl.org/rss/1.0/modules/slash/	19.142%
http://purl.org/syndication/thread/1.0	5.547%
http://a9.com/-/spec/opensearchrss/1.0/	5.487%
http://purl.org/dc/terms/	4.626%
http://backend.userland.com/creativeCommonsRssModule	3.797%
http://www.rawvoice.com/rawvoiceRssModule/	2.901%
http://www.w3.org/2003/01/geo/wgs84_pos#	2.873%
http://www.w3.org/1999/xhtml	1.345%
http://a9.com/-/spec/opensearch/1.1/	0.931%
http://bbc.co.uk/2009/01/ppgRss	0.816%
http://www.thespringbox.com/dtds/thespringbox-1.0.dtd	0.567%
http://webns.net/mvcb/	0.287%
http://radiofrance.fr/Lancelot/Podcast#	0.265%
http://posterous.com/help/rss/1.0	0.201%
http://www.castfire.com/dtds/rss.dtd	0.191%
http://cstv.com	0.147%
http://www.cstv.com	0.147%
http://www.podzinger.com	0.147%
http://api.npr.org/nprml	0.140%
http://blogs.law.harvard.edu/tech/creativeCommonsRssModule	0.131%
http://www.cbsradio.com/	0.131%
http://www.itunesu.com/feed	0.118%
http://libsyn.com/rss-extension	0.102%
http://www.adobe.com/amp/1.0	0.086%
http://pipes.yahoo.com	0.080%
http://www.georss.org/georss	0.070%
http://podfm.ru/RSS/extension	0.064%
http://channel9.msdn.com	0.051%
http://madskills.com/public/xml/rss/module/trackback/	0.048%
http://www.blogger.com/atom/ns#	0.041%
http://purl.org/atom-blog/ns#	0.041%
http://www.rsr.ch/xml/namespace	0.035%
http://boxee.tv/spec/rss/	0.032%

Table 55: The Top 40 Podcast XML Namespaces

Language Code	Feeds %
en	58.500%
en-us	31.945%
en-gb	1.993%
EMPTY FIELD	1.396%
de	1.192%
es	0.797%
fr	0.721%
en-ca	0.450%
de-de	0.386%
en-au	0.325%
en-PI	0.316%
pt-br	0.306%
en-uk	0.281%
fr-FR	0.271%
ja	0.239%
es-mx	0.226%
it-it	0.201%
It	0.188%
es-es	0.179%
Ar	0.156%
zh	0.156%
zh-cn	0.131%
ru	0.124%
ru-ru	0.121%
nl	0.112%
cs	0.108%
de-A T	0.099%
en-ie	0.086%
ko	0.080%
sl	0.080%
en-en	0.070%
es-pr	0.070%
en-nz	0.061%
fr-ch	0.061%
nl-nl	0.057%
ko-kr	0.054%
pt	0.054%
zh-hk	0.054%
zh-tw	0.054%
DA	0.045%

Table 56: The Top 40 Podcast Languages

Generator	Feeds %
EMPTY FIELD	25.642%
Libsyn WebEngine	7.993%
http://podbean.com/?v=3.2	5.484%
podOmatic RSS Generator	4.852%
http://wordpress.org/?v=3.3.2	4.266%
Blogger http://www.blogger.com	4.228%
http://wordpress.org/?v=3.3.1	3.998%
EZ Rss 0.1	3.466%
Blogger	2.155%
http://wordpress.org/?v=3.2.1	1.696%
Castermaster 1.0	1.355%
Podcast Maker v1.4.0 - http://www.lemonzdream.com/podcastmaker	1.253%
http://wordpress.com/	1.237%
Hipcast RSS Feeder 1.25	1.189%
FeedForAll v2.0 (2.0.2.9) http://www.feedforall.com	0.998%
Podcast Maker v1.4.1 - http://www.lemonzdream.com/podcastmaker	0.998%
Podcast Maker v1.3.8b - http://www.lemonzdream.com/podcastmaker	0.937%
FeedForAll Mac v2.1 (2.1.0.1); http://www.FeedForAll.com/	0.861%
PodShow PDN	0.772%
AudioAcrobat RSS Feeder 1.25	0.644%
http://wordpress.org/?v=2.9.2	0.590%
Feeder 1.5.10(880) http://reinventedsoftware.com/feeder/	0.555%
iWeb 3.0.1	0.545%
Loudblog	0.536%
FeedForAll v2.0 (2.0.3.1) http://www.feedforall.com	0.523%
iWeb 3.0.4	0.516%
podcastmachine.com	0.485%
http://wordpress.org/?v=3.0.1	0.453%
http://wordpress.org/?v=	0.421%
iWeb 1.1.2	0.411%
http://wordpress.org/?v=3.1	0.395%
iWeb 2.0.4	0.383%
http://wordpress.org/?v=3.3	0.344%
Podcast Maker v1.3.6 - http://www.lemonzdream.com/podcastmaker	0.332%
http://wordpress.org/?v=3.0.4	0.325%
JellyCast http://www.jellycast.com	0.316%
TypePad http://www.typepad.com/	0.300%
http://wordpress.org/?v=3.1.3	0.290%
Podcast Generator 1.3 - http://podcastgen.sourceforge.net	0.284%
http://wordpress.org/?v=2.7.1	0.281%

Table 57: The Top 40 RSS 2.0 Authoring Tools